

Convolutional Neural Network Models and Interpretability for the Anisotropic Reynolds Stress Tensor in Turbulent One-dimensional Flows

Haitz Sáez de Ocáriz Borde^{*1}, David Sondak^{†2}, and Pavlos Protopapas^{‡2}

¹Department of Aeronautics, Imperial College London, London SW7 2AZ, United Kingdom

²Institute for Applied Computational Science, Harvard University, Cambridge, MA 02138, United States

First Version: March 11, 2021. This version: July 1, 2021

Abstract

The Reynolds-averaged Navier-Stokes (RANS) equations are widely used in turbulence applications. They require accurately modeling the anisotropic Reynolds stress tensor, for which traditional Reynolds stress closure models only yield reliable results in some flow configurations. In the last few years, there has been a surge of work aiming at using data-driven approaches to tackle this problem. The majority of previous work has focused on the development of fully-connected networks for modeling the anisotropic Reynolds stress tensor. In this paper, we expand upon recent work for turbulent channel flow and develop new convolutional neural network (CNN) models that are able to accurately predict the normalized anisotropic Reynolds stress tensor. We apply the new CNN model to a number of one-dimensional turbulent flows. Additionally, we present interpretability techniques that help drive the model design and provide guidance on the model behavior in relation to the underlying physics.

Keywords: turbulence modeling, Reynolds-averaged Navier-Stokes, deep learning, convolutional neural networks, interpretability.

^{*}haitz.saez-de-ochariz-borde17@imperial.ac.uk

[†]dsondak@seas.harvard.edu

[‡]pavlos@seas.harvard.edu

1 Introduction

Most real-world fluid problems and applications involve turbulent flow. As such, engineers and scientists remain concerned about developing reliable turbulence models that have good performance and that at the same time are practical and tractable from a computational perspective. Given that direct numerical simulations (DNS) of turbulent flow are too computationally expensive at the Reynolds numbers of interest, researchers have developed a number of alternative simplified and more practical turbulence models. Two popular approaches are Reynolds-averaged Navier-Stokes (RANS) models and large eddy simulation (LES) models. RANS models are less computationally expensive than LES models, but also less accurate. RANS models aim to determine the average velocity and pressure fields and to model the effects of the fluctuating components on the average. The effect of the fluctuating fields arises due to the Reynolds stress tensor, which must be modelled. Although more traditional two-equation eddy viscosity models such as the $k - \epsilon$ and the $k - \omega$ are commonly used in many engineering applications, their applicability to different flows is limited (see Speziale (1991); Johansson (2002); Chen et al. (2003); Gatski (2004)).

Recently, several researchers have worked on applying machine learning algorithms to develop turbulence closure models for RANS, such as Ling et al. (2016a), Ling et al. (2016b), Fang et al. (2018), Wu et al. (2018), Kaandorp (2018), Song et al. (2019), Kaandorp and Dwight (2020) and Zhu and Dinh (2020). Ling et al. (2016b) proposed the Tensor Basis Neural Network (TBNN) to learn the coefficients of an integrity basis for the Reynolds anisotropy tensor from Pope (1975). The TBNN automatically guarantees the physical invariance in the predicted Reynolds anisotropy tensor. Inspired by Ling et al. (2016b), Fang et al. (2018) used machine learning to model the Reynolds stress for fully-developed turbulent channel flow. They used a generic fully-connected feedforward neural network (FCFF) as the base model and increased its capabilities by using non-local features, directly incorporating friction Reynolds number information, and enforcing the boundary condition at the channel wall. A survey regarding recent work in data-driven turbulence modeling can be found in Duraisamy et al. (2019).

In this paper, we present a convolutional neural network (CNN) architecture that outperforms the fully-connected architectures developed by Fang et al. (2018) for the one-dimensional turbulent channel flow problem. We train and test our new architecture on one-dimensional turbulent flow configurations and demonstrate that the new neural network works beyond the simple channel flow. We also propose and conduct several interpretability analyses to identify significant regions of the flow and guide the architecture design.

Although an exact definition for interpretability has not been developed yet, it could be described as the ability to explain in understandable terms the choices made by a machine learning algorithm to a human. Interpretability has recently gathered a lot of attention in the machine learning community. See Zhang et al. (2021) for a survey on some of the latest interpretability techniques. Other less recent surveys on interpretability are those by Carvalho et al. (2019) and by Chakraborty et al. (2017). There is a growing interest in understanding how machine learning models make decisions and how these relate to the underlying problem. It is unlikely that machine learning augmented turbulence models will be ultimately successful without the need for direct human expertise and hence, understanding the algorithm decision making process is key to improving the models. Most of the work on interpretability so far has been quite general, such as that by Molnar (2019), Gilpin et al. (2019) and Doshi-Velez and Kim (2017). These works serve as a good introduction to interpretability, but only present generic algorithms that do not directly address interpretability in a turbulence modelling context. To the best of our knowledge, interpretability techniques have not been applied to machine learning in turbulence modeling to guide architecture design. In this paper, we will try to address this issue and adapt some interpretability techniques used in other machine learning applications to our turbulence modelling problem.

The rest of the paper has the following structure. Section 2 covers background information

on the RANS equations, several one-dimensional turbulent flows, the datasets used, and neural networks and interpretability. Section 3 reviews the neural network models by Fang et al. (2018) and introduces our new convolutional models. Section 4 describes the interpretability techniques that have been adapted for our turbulence modelling problem. Section 5 covers the performance of the different machine learning models and presents the interpretability results. Lastly, Section 6 summarizes the final conclusions and suggests future work directions.

2 Background and Methodology

The present work aims at developing CNN models to represent the Reynolds anisotropy tensor for turbulent one-dimensional flows, as well as at applying interpretability techniques to understand how the models make decisions and how these relate to the underlying physics of the problem. In this section, some background on turbulent channel flow, turbulent Couette flow, and turbulent channel flow with wall transpiration, neural networks, and interpretability are provided.

2.1 The RANS Equations

The Reynolds-averaged approach decomposes the velocity into average and fluctuating components $\mathbf{u} = \bar{\mathbf{u}} + \mathbf{u}'$ to find the average field $\bar{\mathbf{u}}$. This averaging operation applied on the Navier-Stokes equations leads to the RANS equations. The RANS equations include an additional term $\overline{\mathbf{u}' \otimes \mathbf{u}'}$ known as the Reynolds stress tensor, and this must be modelled to close the RANS equations. A substantial portion of modelling efforts have concentrated on the anisotropic Reynolds stress tensor $\mathbf{a} = \overline{\mathbf{u}' \otimes \mathbf{u}'} - (2k/3)\mathbf{I}$ as that is the portion responsible for turbulent transport, where k is the turbulent kinetic energy $k = \frac{1}{2}\text{trace}(\overline{\mathbf{u}' \otimes \mathbf{u}'})$. In this work, the models are trained on the normalized anisotropy tensor, $\mathbf{b} = \mathbf{a}/(2k)$, and individual components of the tensor are denoted with subscripts referring to the fluctuating velocity components involved (e.g. $b_{uv} = \overline{u'v'}/(2k)$).

Eddy viscosity models such as the $k - \epsilon$ and the $k - \omega$ models are some of the most popular models used to obtain predictions for the anisotropic Reynolds stress tensor, but they do not provide satisfactory results for all flows (see e.g. Speziale (1991); Johansson (2002); Gatski (2004); Chen et al. (2003)). Recently, machine learning algorithms have been proposed to tackle this problem and obtain predictions for the anisotropic Reynolds stress tensor in a variety of flows, including those for which eddy viscosity models are inadequate (see e.g. Tracey et al. (2015); Zhang and Duraisamy (2015); Ling et al. (2016b); Wu et al. (2018); Wang et al. (2017)). In particular, the TBNN by Ling et al. (2016b) has provided improved predicative accuracy compared to traditional RANS models when trained and tested on several flows.

2.2 Turbulent Channel Flow

Turbulent channel flow is a paradigmatic flow in which a fluid confined between two parallel plates in the $x - z$ plane is driven by a pressure gradient in the streamwise direction. The streamwise, spanwise, and wall-normal directions are denoted by x , z , and y , respectively. The goal is to determine the velocity field $\mathbf{u} = (u, v, w)$, where u , v , and w are the velocities in the x , y , and z directions. In general, each component of the velocity field depends on space (x, y, z) and time t . The friction velocity $u_\tau = \sqrt{\tau_{\text{wall}}/\rho}$ is a natural velocity scale for this problem, where τ_{wall} is the wall shear stress and ρ is the fluid density. Selecting the channel half-width h and the friction velocity as the length and velocity scales of the problem leads to the friction Reynolds number $Re_\tau = u_\tau h/\nu$ as a natural Reynolds number for this flow, where ν is the kinematic viscosity of the fluid. Results presented in this paper are typically shown in wall-units, $y^+ = u_\tau y/\nu$. Turbulent channel flow is statistically one-dimensional, its mean velocity field being $\bar{\mathbf{u}} = (\bar{u}(y), 0, 0)$. In this case, the axial momentum equation is solved for $\bar{u}(y)$ and the only component of \mathbf{a} affecting the velocity profile is a_{uv} .

The machine learning models in this work are trained and tested on the same turbulent channel flow dataset used by Fang et al. (2018), which consists of DNS data at four friction Reynolds numbers $Re_\tau = [550, 1000, 2000, 5200]$. The number of DNS data points varies for different friction Reynolds numbers due to differing mesh resolution requirements. For $Re_\tau = [550, 1000, 2000, 5200]$ the dataset has $N_y = [192, 256, 384, 768]$ respective points in the wall-normal direction. The data is accessible at the Oden Institute turbulence file server¹. Detailed information on the DNS simulations can be found in Lee and Moser (2015).

2.3 Turbulent Couette Flow

Turbulent Couette flow is a shear-driven fluid flow in which the fluid is confined between two parallel surfaces one of which is moving tangentially relative to the other. This flow configuration is a proxy for some practical problems, namely, the Earth's mantle movement or extrusion in manufacturing processes. In our case, we consider the flow between two parallel planes with no pressure gradient. Similarly to turbulent channel flow, our turbulent Couette flow configuration is also statistically one-dimensional with a mean velocity field $\bar{\mathbf{u}} = (\bar{u}(y), 0, 0)$, and to find the solutions to the RANS equations \bar{u} must be computed, which depends on a_{uv} . For this flow, the friction Reynolds number Re_τ is defined based on the friction velocity and the channel half-width, as suggested by Lee and Moser (2018), to remain consistent with other studies regarding channel flow, such as those by Lozano-Durán and Jiménez (2014) and Lee and Moser (2015). The motivation for using this definition of Re_τ is to ease comparison between dissimilar flows. Nevertheless, Barkley and Tuckerman (2007) suggested that the appropriate length scale in the case of Couette flow should correspond to the full width according to their study of transitional flows.

The study of planar Couette flow has been more limited in total number of studies and the variety of Reynolds numbers investigated compared to channel flow, both from a experimental and computational perspective. Lee and Moser (2018) attributed this fact to the existence of very-large-scale motions in Couette flow, which is hard to represent in DNS simulations, and tried to address this problem in their paper. We used the DNS data by Lee and Moser (2018) to train and test our model in Couette Flow. The data consisted of three friction Reynolds numbers $Re_\tau = [93, 220, 500]$ with $N_y = [64, 96, 128]$ points in the wall-normal direction. The data for turbulent Couette flow is also accessible at the Oden Institute turbulence file server.

2.4 Turbulent Channel Flow with Wall Transpiration

In addition to the two paradigmatic flows discussed above, we also trained and tested our model on fully-developed turbulent channel flow with wall transpiration. In this case, uniform blowing and suction was used on the lower and upper walls, respectively. It is well known that in the classical plane turbulent channel flow without transpiration, all statistical quantities of the flow are either symmetric or antisymmetric with respect to the channel centerline. However, when wall transpiration is included, these symmetries do not hold, which modifies the scaling laws for the linear viscous sublayer and the law of the wall for wall-bounded flows with no transpiration as discussed by Avsarkisov et al. (2014). This makes turbulent channel flow with wall transpiration an interesting flow case to train and test our model, which will need to be able to capture substantially more complicated flow features than those for channel and Couette flow. For this flow, the friction velocity u_τ is defined as the mean friction velocity $u_\tau = \sqrt{\frac{u_{\tau b}^2 + u_{\tau s}^2}{2}}$, where subscripts b and s denote variables on the blowing and the suction side. Once more, in the case of turbulent channel flow with wall transpiration, \bar{u} must be found to solve the RANS equations, which is only dependent on a_{uv} .

¹<https://turbulence.oden.utexas.edu/>

The DNS data used for turbulent channel flow with wall transpiration had three friction Reynolds numbers $Re_\tau = [250, 480, 850]$ with $N_y = [251, 385, 471]$ respective points in the wall-normal direction. The data is available online at the TU Darmstadt Fluid Dynamics Turbulence DNS data base². Note that the data is provided at a number of $\frac{V_0}{u_\tau}$ ratios, where V_0 refers to the transpiration velocity. In our case, we use the data for $\frac{V_0}{u_\tau} = 5$. For the DNS simulation, [Avsarkisov et al. \(2014\)](#) implemented a numerical code developed at the School of Aeronautics, Technical University of Madrid, which is described in detail by [Hoyas and Jiménez \(2006\)](#).

2.5 Neural Networks

Deep feedforward networks, also called feedforward neural networks or multilayer perceptrons (MLPs), are considered the archetype of deep learning (DL) models. Feedforward networks are used to obtain an approximation of some function f . For instance, $\mathbf{y} = f(\mathbf{x})$ maps an input \mathbf{x} to a prediction \mathbf{y} . A feedforward network defines a mapping $\mathbf{y} = f^*(\mathbf{x}; \boldsymbol{\theta})$ and is optimized to find the parameters $\boldsymbol{\theta}$ that give the best approximation of the function f . In feedforward models the information flows through the function being evaluated from \mathbf{x} , through the intermediate computations, and finally to the output \mathbf{y} , as described by [Goodfellow et al. \(2016\)](#).

Neural networks can usually be represented by composing together several functions and they can be associated with a directed acyclic graph. We may think of $f^*(\mathbf{x})$ as being the connection of n different functions in a chain, so that:

$$f^*(\mathbf{x}) = f^{(n)}(\dots(f^{(2)}(f^{(1)}(\mathbf{x})))), \quad (2.1)$$

where $f^{(1)}, f^{(2)}, \dots$ and $f^{(n)}$ are different functions that would correspond to each layer of the network. The depth of the model corresponds to the overall length of the aforementioned chain. We call the first and last model layers the input and output layers, respectively, and the rest of the layers in between the hidden layers. Furthermore, some authors may refer to the dimensionality of the hidden layers as the width of the model.

The edges connecting two layers are characterized by the weights, \mathbf{w} , and biases, \mathbf{b} , which are trainable parameters that are optimized during the training process. The output of the previous layer, h_i , is transformed using the affine transformation $wh_i + b$. Each node in the layer applies a non-linear operation to the affine transformation so that the output of layer $i + 1$ is $h_{i+1} = \sigma(wh_i + b)$. The non-linear function σ is usually referred to as the activation function which must be smooth. It is worth mentioning that the output layer typically takes special activation functions, different to the ones used in the hidden layers, depending on the type of problem being solved. The model output is the predicted function that is parameterized by the weights and biases in the network.

In the case of a FCFF, the output of a given layer i is passed to layer $i + 1$ and every node in the current layer is connected to every node in the previous and subsequent layer, without any jump connections or feedback loops. On the other hand, CNNs use convolutional layers where each neuron is only connected to a few nearby neurons in the previous layer. Regarding convolutional layers, the same set of weights are shared by every neuron. This connection pattern is applied to cases where the data can be interpreted as being spatial. The convolutional layer's parameters consist of a set of learnable filters or kernels. Every filter has relatively small dimensions but extends through the full depth of the input volume. A convolution is computed by sliding the filter over the output of the previous layer. At every location, a matrix multiplication is performed and sums the result onto the feature map. The feature map is the output of the filter applied to the previous layer. In general, CNNs may also have pooling layers and fully-connected layers before the output. See [Millstein \(2018\)](#) for an introduction to CNNs.

For both FCFFs and CNNs, the model predictions are compared to data in a loss function, and an optimization algorithm (see [Bottou et al. \(2016\)](#)), such as stochastic gradient descent

²https://www.fdy.tu-darmstadt.de/fdyresearch/dns/direkte_numerische_simulation.en.jsp

(see Kiefer and Wolfowitz (1952) and Ruder (2016)), is used to adjust all the weights and biases, θ , of the network to minimize the loss function. The process of optimizing a neural network can be challenging since it is a high-dimensional non-convex optimization problem.

Adopting neural networks for physics-based problems entails the challenge of trying to inform the network with known physical laws, since otherwise the network will simply try to memorize the patterns in the data without paying attention to the underlying physics. One of the first attempts to embed the physical and mathematical structure of the Reynolds anisotropy tensor into a neural network was the TBNN by Ling et al. (2016a). The TBNN guaranteed Galilean and rotational invariance of the predicted Reynolds anisotropy tensor by adding an additional tensorial layer to a FCFF network which returned the most general, local eddy viscosity model described in Pope (1975). Fang et al. (2018) further proposed a number of techniques such as reparameterizing a FCFF network to enforce the no-slip boundary condition as a function of the normalized distance from the wall, explicitly providing Re_τ to the network to incorporate friction Reynolds number information into the model, and extensions to allow for non-locality. In the present work, some of the techniques by Fang et al. (2018) will be applied to CNNs.

2.6 Interpretability

One of the biggest problems regarding artificial intelligence's (AI) public trust and acceptance is that most algorithms exhibit what is known as a black box behaviour. A black box algorithm lets the user see the input and output, but it gives no view of the learning process and does not explicitly share how and why the model reaches a given conclusion. Particularly in the case of DL, there may be a great number of hidden layers which allow the algorithm to automatically determine complex patterns in the input data but which obscure the inner functioning of the network and the relationship between input and output. Interpretability of neural networks aims at addressing this problem by shedding some light on the behaviour of the algorithms.

Understanding how algorithms work and make decisions is of paramount importance if we want to be able to rely on AI to make economical decisions, manage public health or, in the case of fluid mechanics, understand how our algorithm relates to the underlying physics problem that we are trying to solve. Interpretability is also a powerful tool that can be used to identify flaws in the model and improve its performance. It is currently a field in development and as mentioned by Molnar (2019), researchers have not even agreed on a proper definition for it yet. Some non-mathematical definitions have been proposed, such as those by Miller (2019): “Interpretability is the degree to which a human can understand the cause of a decision.” and Kim et al. (2016): “Interpretability is the degree to which a human can consistently predict the model’s result.”

In this work, a number of interpretability techniques will be explored to understand the trained models and improve their architecture, explain their performance, and find relations between the model decisions and the underlying physics. Using different interpretability methods will help us validate our results.

3 Neural Network Models

In this section, we review the FCFF neural network architectures proposed by Fang et al. (2018) to model the Reynolds anisotropy tensor for turbulent channel flow and introduce new CNN models. The following non-dimensional quantities are used by all models: the non-dimensional distance from the wall $y^+ = \frac{u_\tau y}{\nu}$, where u_τ is the friction velocity and ν the kinetic viscosity; the normalized mean velocity gradient $\frac{d\bar{u}}{dy}^* = \frac{d\bar{u}}{dy} \frac{k}{\epsilon}$, where k is the turbulent kinetic energy and ϵ is the turbulence dissipation rate; the normalized Reynolds anisotropy tensor, which is given by $\mathbf{b} = \frac{\mathbf{a}}{2k}$ and the friction Reynolds number, Re_τ . In this work, the input to the models is the normalized mean velocity gradient of the flow $\frac{d\bar{u}}{dy}^*$ and the output of the model is the $u - v$ component of the normalized Reynolds anisotropy tensor b_{uv} .

3.1 Review of Fully-Connected Feedforward Models for Turbulent Channel Flow

Fang et al. (2018) proposed a generic FCFF model as the simple baseline neural network model for their work and suggested three modifications to embed the physics of the turbulent channel flow problem into the network: to enforce the boundary condition $b_{uv} = 0$ at the wall through a reparameterization, explicitly providing Re_τ to the network, and extensions to allow for non-local models. The present work will focus on the implementation of the first two techniques which proved to give the best results for turbulent channel flow.

The baseline FCFF model is rather simple and has no embedded physics other than Galilean invariance since it uses the normalized mean velocity gradients as input. Figure 1a shows a schematic representation of the baseline model, which takes the normalized mean velocity gradient at a given distance from the wall and predicts the $u - v$ component of the normalized Reynolds anisotropy tensor at that location. Note that both input and output are scalars.

The first improvement that Fang et al. (2018) proposed for the baseline model was enforcing the no-slip boundary condition at the channel wall by reparameterizing the output as follows,

$$b_{uv} \left(\frac{d\bar{u}}{dy}^*, y^+ \right) = A(y^+) \text{FCFF} \left(\frac{d\bar{u}}{dy}^* \right), \quad (3.1)$$

where $\text{FCFF} \left(\frac{d\bar{u}}{dy}^* \right)$ is the output of the baseline model and $A(y^+)$ is a user-selected function that must satisfy the condition $A(0) = 0$. The function $A(y^+) = 1 - e^{-\beta y^+}$ with hyperparameter β was suggested in the original paper, which guarantees that the solution satisfies the condition $b_{uv} = 0$ at $y^+ = 0$. The improved model takes y^+ as an additional input and the loss function is calculated based on the final reparameterized solution. Figure 1b displays a diagram of the reparameterized model with the boundary condition enforcement.

Fang et al. (2018) also suggested inputting the friction Reynolds number, Re_τ , of the channel flow directly into the model as the friction Reynolds number has a direct influence on the mean velocity profile and, therefore, on the anisotropic Reynolds stress tensor. Re_τ may be fed into one or more of the intermediate layers of the FCFF model as an additional input, as shown in Figure 1c. In practice, the difference between inputting the friction Reynolds number in different layers was deemed insignificant. Furthermore, combining both boundary condition enforcement and friction Reynolds number injection as shown in Figure 1d was found to give even better results.

3.2 Convolutional Neural Network Models

In this work, we present new one-dimensional CNN models that outperform the original FCFF models previously described for turbulent channel flow. We propose a new CNN baseline model which is fully convolutional, without pooling layers, and a fully-connected region. The proposed model, which we describe in more detail in Section 5.1, has multiple layers and multiple filters. The CNN model takes as input an array with all the normalized mean velocity gradient values along the height of the channel for a given friction Reynolds number and predicts all the normalized anisotropic Reynolds stress tensor values for that friction Reynolds number simultaneously. This is unlike the FCFF model which takes a single value of the normalized mean velocity gradient and makes a single prediction at a time. Padding is used to preserve the original length of the input array throughout all the convolutional layers, and batch normalization helps the network to train faster and be more likely to be stable (see Ioffe and Szegedy (2015)). The final convolutional layer produces several activations with the same length as the original input array. The weighted sum of the activations is calculated to obtain the final output as shown in Figure 2. Note that the baseline CNN model also guarantees Galilean invariance since it uses the normalized mean velocity gradients as inputs. Figure 3 shows the inner architecture of the CNN baseline model.

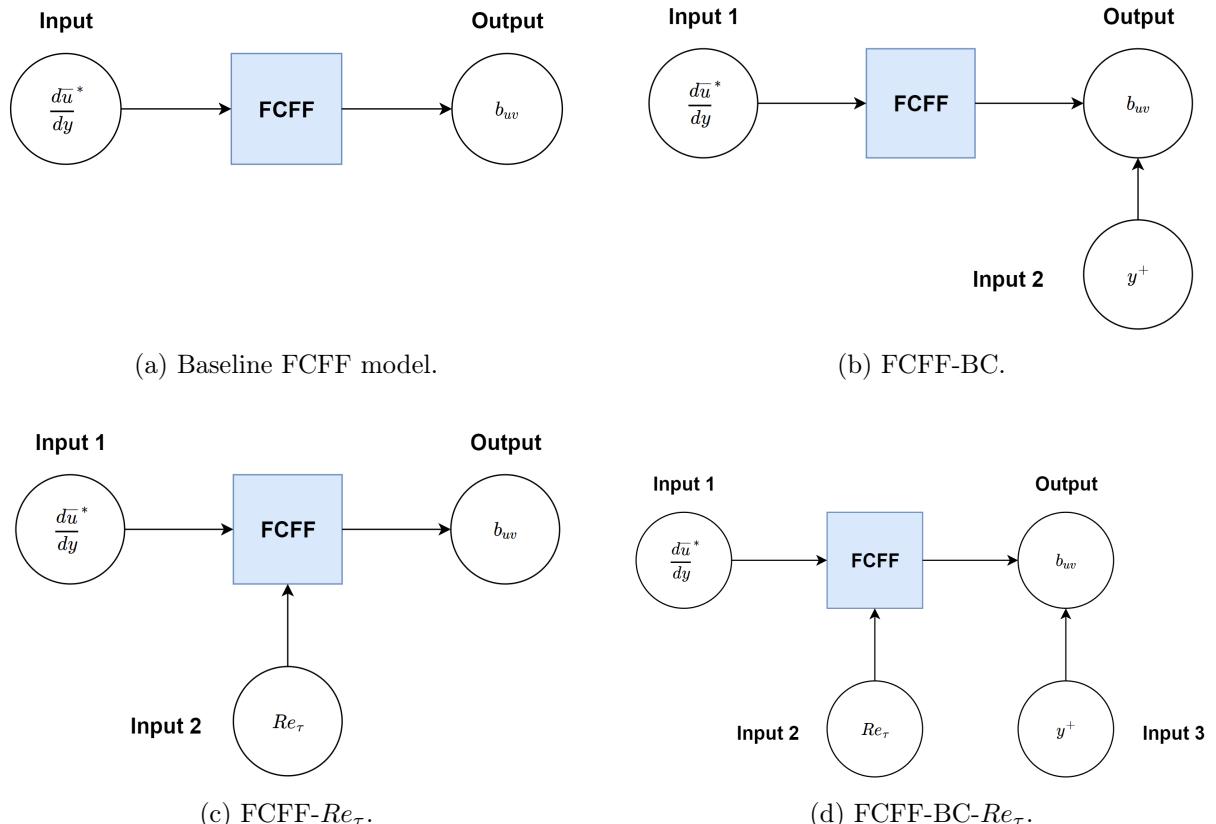


Figure 1: Diagrams of FCFF models for turbulent channel flow. (a) Baseline FCFF model. (b) FCFF model with boundary condition enforcement. (c) FCFF model with friction Reynolds number injection. (d) FCFF model with boundary condition enforcement and friction Reynolds number injection.

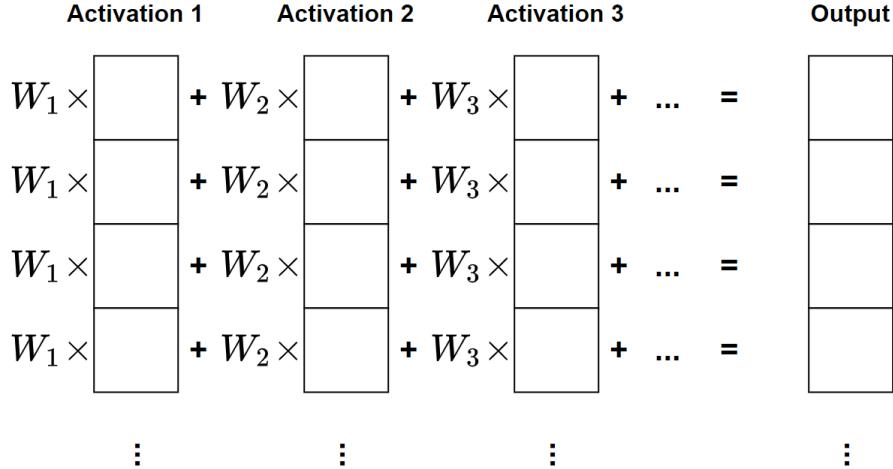


Figure 2: Diagram of the final weighted sum operation. Each activation is multiplied by a trainable weight.

The no-slip boundary condition at the channel wall can also be enforced using the same approach as with the FCFF model,

$$\mathbf{b}_{uv} \left(\frac{d\bar{u}^*}{dy}, \mathbf{y}^+ \right) = A(\mathbf{y}^+) \text{CNN} \left(\frac{d\bar{u}^*}{dy} \right), \quad (3.2)$$

where $\text{CNN} \left(\frac{d\bar{u}^*}{dy} \right)$ is the output of the baseline CNN model, and \mathbf{y}^+ and $A(\mathbf{y}^+)$ are the distance from the wall in viscous units and the user-selected function as described before in Section 3.1. Figure 4b shows the diagram of the CNN model with boundary condition enforcement.

To input the friction Reynolds number into the CNN model we concatenate an array with the friction Reynolds number to the original input array with the normalized velocity gradients. The new input is a tensor with two channels, one with the non-dimensionalized velocity gradients evaluated at all the points at different locations along the height of the channel for a given friction Reynolds number, and the other with the friction Reynolds number information and all entries in the second channel being equal to the friction Reynolds number, as shown in Figure 4c. As in the case of the FCFF models, boundary condition enforcement and friction Reynolds number injection can simultaneously be applied to the CNN model (Figure 4d).

4 Interpretability Techniques

In the context of turbulence modelling, developing interpretability techniques will help better understand the relation between the model decisions and the underlying physics of the problem as well as to improve the model architecture. In this section, we will discuss a number of interpretability techniques that we will employ to explain and improve our CNN turbulence models.

4.1 Activation Visualization

Activation visualization is a straightforward technique that gives some insight into how the layer structure of a CNN model processes the data. As discussed by Chollet (2018), activation visualization consists of displaying the feature maps that are output by each layer of the model to visualize how successive convolutional layers apply transformations to the input, giving the user a high-level understanding of the meaning of each individual convolutional filter. The output of

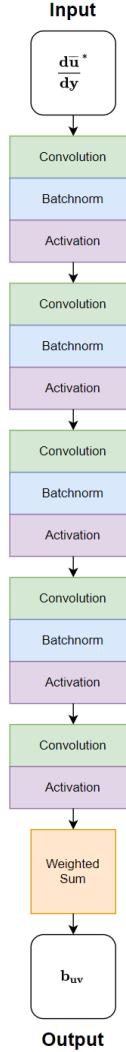


Figure 3: Diagram of the inner architecture of the baseline CNN model for turbulent one-dimensional flows. The input is an array for the normalized velocity gradient profile evaluated at different locations along the height of the channel for a given friction Reynolds number. The output is an array with the corresponding predicted normalized anisotropy tensor values. The one-dimensional convolutional layers are shown in green, the one-dimensional batch normalization layers with trainable parameters are shown in blue, the activation functions in purple and the final weighted sum operation across the channel dimension in orange.

each filter may to some degree be attributed to visual or physical representations of the input data that the model is trying to capture.

In this paper, we show how to use activation visualization as a neural network design driver for our one-dimensional regression problem (see Section 5.4.1 for examples). By looking at the activations of the different convolutional layers within a network it is possible to determine whether some of the filters are redundant (because they are not capturing any new relationship) or more are needed (if we see the layer is struggling to learn a smooth mapping between the input and the output). In this way we can ease common issues such as underfitting and overfitting.

4.2 Occlusion Sensitivity

Occlusion sensitivity was first explored by authors such as [Zeiler and Fergus \(2013\)](#) in the image classification context. In their work, the authors tried to evaluate whether the machine learning

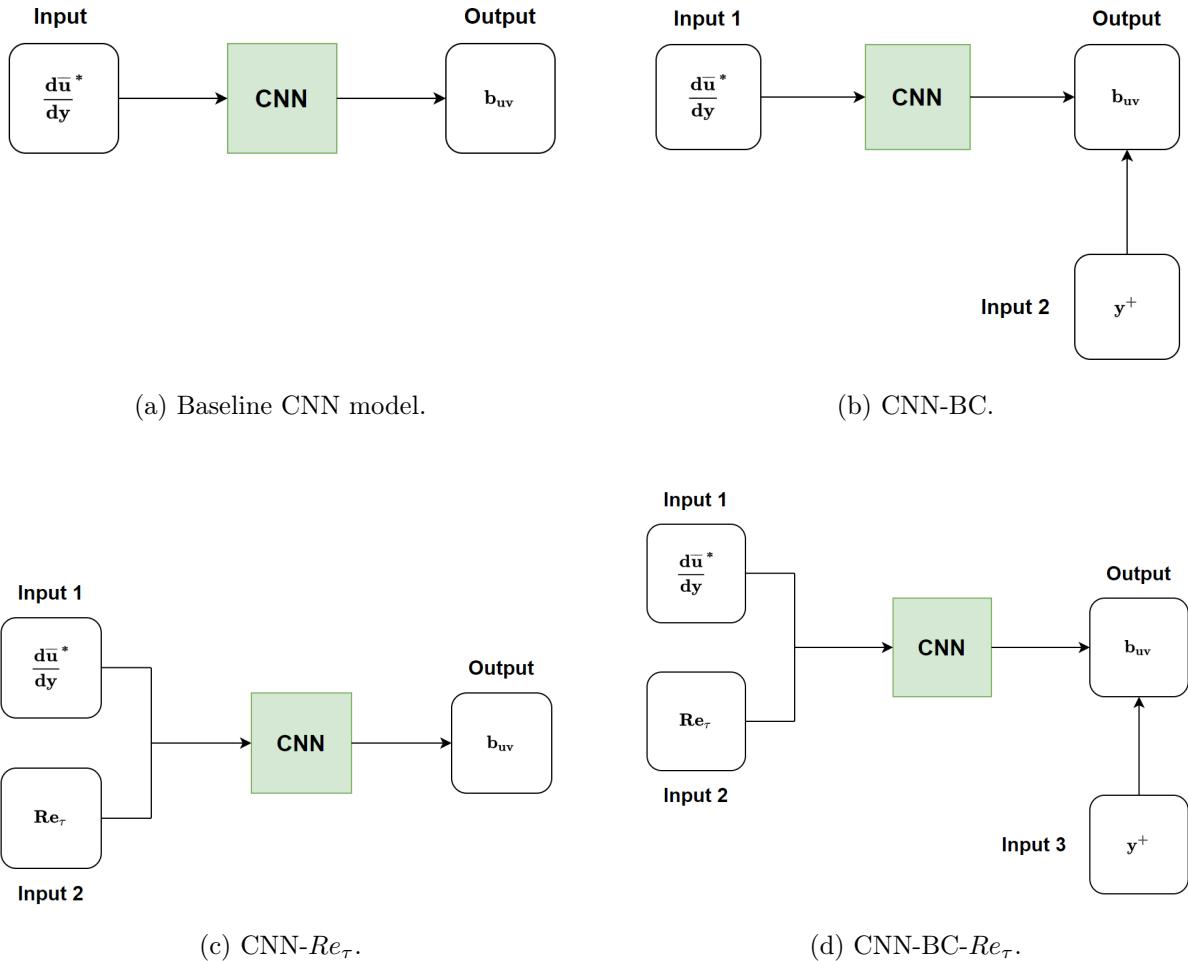


Figure 4: Diagrams of CNN models for turbulent one-dimensional flows. (a) Baseline CNN model. (b) CNN model with boundary condition enforcement. (c) CNN model with friction Reynolds number injection. (d) CNN model with boundary condition enforcement and friction Reynolds number injection.

model was genuinely identifying the position of the relevant object in the image, or if it was merely paying attention to the surrounding context instead. To determine this question, they occluded different portions of the input image by sliding a grey square over it and monitored the output of the classifier. [Ancona et al. \(2017\)](#) performed similar experiments to determine which parts of an input cause a machine learning model to change its outputs the most.

We extend this technique to our one-dimensional regression problem. We systematically occlude the input array by setting the value of the mean velocity gradients to zero at specific regions along the channel height using a sliding window and evaluate how this affects the accuracy of the prediction. We construct two different types of plots: 1.) one-dimensional global occlusion sensitivity plots that evaluate the effect of occluding different regions of the input on the overall loss of the model output and 2.) two-dimensional local occlusion sensitivity plots that aim at describing the impact of occlusion on the individual accuracy of all entries of the output array. To evaluate the loss in accuracy for the global occlusion sensitivity plots, we look at the absolute difference between the original loss of prediction and the loss of the model output when the input is partially occluded,

$$\mathcal{D}_g = |L(\mathbf{y}^{\text{true}}, \mathbf{y}) - L(\mathbf{y}^{\text{true}}, \mathbf{y}^{\text{occlusion}})|, \quad (4.1)$$

where \mathbf{y}^{true} is the normalized anisotropic Reynolds stress tensor across the entire domain. For the local occlusion sensitivity plot, we individually evaluate the absolute difference of the prediction accuracy of a single output entry as follows:

$$\mathcal{D}_l = |L(y_i^{\text{true}}, y_i) - L(y_i^{\text{true}}, y_i^{\text{occlusion}})|. \quad (4.2)$$

In (4.1) and (4.2), L is the loss function used to train the algorithm. In (4.2), y_i^{true} corresponds to the normalized anisotropic Reynolds stress tensor value at a given position along the channel height, y_i is the original model prediction for the anisotropic Reynolds stress tensor at that position, and $y_i^{\text{occlusion}}$ refers to the prediction when some of the input entries are occluded. Based on these metrics we are able to evaluate which portions of the input array are most important to preserve the accuracy of the original prediction by looking at the global loss and local loss difference when we occlude regions of the input.

4.3 Gradient-based Sensitivity

[Simonyan et al. \(2014\)](#) introduced the first gradient-based saliency map for image classification. This approach is often referred to as the vanilla gradient-based saliency map, which they obtained by computing the gradient of the class score with respect to the input image. For image classification, the intuition is that by calculating the change in the predicted class by applying small perturbations to the pixel values across the image, it is possible to measure the relative importance of each pixel to the final model prediction.

In our context, we calculate the absolute value of the partial derivative of the loss. The gradient provides a convenient mechanism to determine the sensitivity of the output with respect to the input. However, the main problem with gradient-based sensitivity maps in the case of image classification and regression is that they tend to be noisy.

Researchers have hypothesized about the apparent noise in raw gradient visualization and suggested that using the raw gradients as a measure for feature importance is not optimal. [Smilkov et al. \(2017\)](#) argued that the noise present in sensitivity maps may arise due to meaningless local variations in partial derivatives, especially given that many modern DL models use ReLU activation functions or other similar functions like ELU or Leaky ReLU. Such functions are not continuously differentiable which may lead to abrupt changes in the gradients. Furthermore, [Smilkov et al. \(2017\)](#) showed that applying a Gaussian kernel to the gradients gives more interpretable results that filter out some of the noise.

Note that in the case of gradient-based sensitivity we can also calculate the global sensitivity by calculating the partial derivative of the overall loss with respect to the input,

$$\mathcal{G}_g = \left| \frac{\partial L(\mathbf{y}^{\text{true}}, \mathbf{y})}{\partial \mathbf{x}} \right|, \quad (4.3)$$

or the local sensitivity by calculating the partial derivative of the loss of an individual entry of the output with respect to the input,

$$\mathcal{G}_l = \left| \frac{\partial L(y_i^{\text{true}}, y_i)}{\partial \mathbf{x}} \right|. \quad (4.4)$$

However, results obtained using (4.4) tend to be too noisy and difficult to interpret, so we therefore use the global gradients in this work.

5 Results

In this section, we first discuss the model results and compare the performance of the original FCFF models by [Fang et al. \(2018\)](#) with the new CNN models. Next, we apply the interpretability techniques described in Section 4 to the CNN models.

5.1 Model Results for Turbulent Channel Flow

Table 1 displays the four training prediction cases that are examined for turbulent channel flow. The models are trained on the data for three friction Reynolds numbers and tested on the data for the remaining friction Reynolds number. Although the same dataset is used to train both the baseline FCFF model and the baseline CNN, the training procedure is not the same for the two types of models as they are inherently different. The FCFF model by [Fang et al. \(2018\)](#) maps a single scalar input to a scalar output,

$$b_{uv} = \text{FCFF} \left(\frac{d\bar{u}^*}{dy} \right), \quad (5.1)$$

whereas the CNN model maps an input array to an output array,

$$\mathbf{b}_{uv} = \text{CNN} \left(\frac{d\bar{\mathbf{u}}^*}{d\mathbf{y}} \right). \quad (5.2)$$

The FCFF model implicitly assumes that the normalized Reynolds stress tensor is only dependent on the local normalized mean velocity gradient, which we know to be false. [Fang et al. \(2018\)](#) tried to address this problem by introducing non-locality using fractional derivatives but it was not as satisfactory as expected. On the other hand, because the CNN model proposed in the present work takes the whole normalized mean velocity gradient as input, it is more effective at capturing non-local relationships.

Table 1: The four training-prediction cases for turbulent channel flow.

Case	Training set	Test set
1	$Re_\tau = [550, 1000, 2000]$	$Re_\tau = 5200$
2	$Re_\tau = [550, 1000, 5200]$	$Re_\tau = 2000$
3	$Re_\tau = [550, 2000, 5200]$	$Re_\tau = 1000$
4	$Re_\tau = [1000, 2000, 5200]$	$Re_\tau = 550$

All FCFF models had 5 layers with 50 nodes each and ELU activation functions as in the paper by Fang et al. (2018). To train the FCFF models we randomly split and shuffled the training data into 80% training and 20% validation data. We empirically found that training the model to predict ten times the normalized anisotropic Reynolds stress tensor, $10b_{uv}$, helped with model convergence, and we simply divide the model output of the trained model by 10 to make predictions. The mean squared error between the labels and the predicted value of the normalized anisotropic Reynolds stress tensor was used as the loss function. The training optimization was performed using the Adam optimizer Kingma and Ba (2014) with an initial learning rate of 10^{-4} and a batch size of 10. The weights and biases of the networks were initialized using the Kaiming He uniform distribution as described in He et al. (2015). To prevent over-fitting we used early stopping by monitoring the validation loss as suggested by Prechelt (1998). For the boundary condition enforcement function $A(y^+) = 1 - e^{-\beta y^+}$ we found $\beta = 0.1$ to work best. For friction Reynolds number injection, multiplying Re_τ times 10^{-8} helped the training. The relative order of magnitude of the input is of paramount importance when training a DL algorithm for fast convergence. We found that this scaling helped substantially speed up the optimization, allowing the neural network to capture information about the friction Reynolds number while still focusing most of its attention on the velocity gradients. Figure 5 shows the training and validation loss during training of the FCFF-BC- Re_τ model (FCFF model with boundary condition enforcement and friction Reynolds number injection) for Case 2 as a function of the number of epochs. In this case, training took 6577 epochs. For the rest of the FCFF models the number of epochs was of the same order of magnitude.

On the other hand, the training procedure for the CNN models was modified with respect to the FCFF models. We considered the training data as three “images” of the normalized mean velocity gradient profile, one for each of the friction Reynolds numbers. The three images were repeatedly fed into the model. The batch size was 1 since we fed one image to the model at a time. Given that the number of data points for different friction Reynolds numbers varies in the DNS dataset —recall from Section 2.2 that $Ny = [192, 256, 384, 768]$ for $Re_\tau = [550, 1000, 2000, 5200]$ — we used padding on the profiles with less data points to equalize the length of all inputs. Again, we found that training the model to predict ten times the normalized anisotropic Reynolds stress tensor, $10b_{uv}$, helped with model convergence. The mean squared error between the labels and the predicted value of the normalized anisotropic Reynolds stress tensor was used as the loss function for the CNN models as well. However, note that instead of evaluating the loss on a single prediction, the mean squared error of the whole \mathbf{b}_{uv} profile prediction for a given friction Reynolds number was evaluated all together. We also used the Adam optimizer with initial learning rate of 10^{-4} and the Kaiming He initialization for the convolutional layers. The CNN model had 5 convolutional layers and the first four included batch normalization with trainable parameters before applying the ELU activation function as shown in Figure 3. The first convolutional layer had 5 filters of kernel size 3, the second layer had 5 filter of kernel size 11, the third layer had 10 filters of kernel size 31 and the last two convolutional layers had 10 filters of kernel size 41. Given that the last convolutional layer had 10 filters, 10 weights were used in the final weighted sum. The trainable parameters for the weighted sum at the end of the model architecture were initialized at $-\frac{1}{10}$. The negative sign in the weights improved training and eased the mapping from input to output. We empirically found that using big kernel sizes in the last layers guaranteed a smooth output curve. Additionally, weight decay with hyperparameter $\lambda = 10^{-4}$ (found through hyperparameter optimization) was used and observed to help obtain a smooth solution without significantly slowing down training. The boundary condition enforcement hyperparameter β remained unaltered from the FCFF model. Once again, multiplying the friction Reynolds number times 10^{-8} before injection also helped the training of this model. In the case of the CNN models, training lasted 10000 to 25000 epochs for turbulent channel flow, significantly longer than in the case of the FCFF model. Alternatively, one may choose to use a deeper neural network with smaller kernel sizes. We found that to obtain

similar results using a maximum kernel size of 11 the neural network had to be at least twice as deep which slowed down training. Using approximately the same number of trainable parameters in a deeper network with smaller kernel sizes implied that the algorithm had to calculate longer chain rules to update the trainable parameters even as the model was more likely to suffer from vanishing gradient problems.

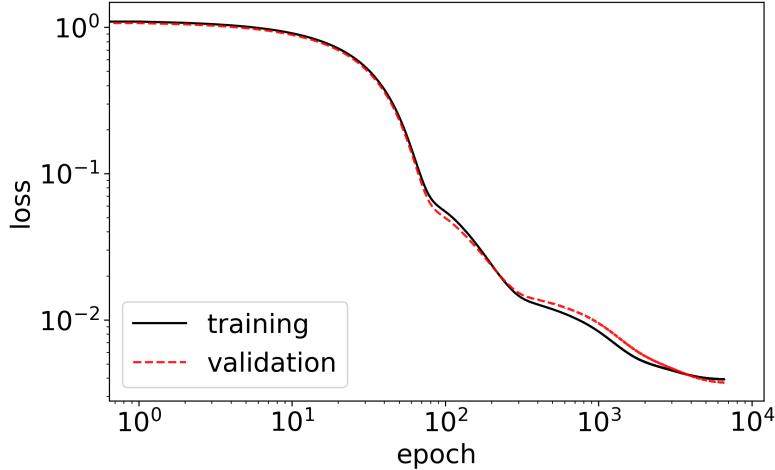


Figure 5: The training and validation loss as a function of the number of epochs during training for the FCFF-BC- Re_τ model for Case 2. The model converged after 6577 epochs.

For model evaluation, we used the R^2 score, which is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable in a regression model. An R^2 of 1 indicates a perfect fit. The R^2 can be negative on the validation set if the regression does worse than the sample mean in terms of tracking the dependent variable. The R^2 scores of the various models on the training and test data are displayed in Table 2.

The CNN models consistently outperform the FCFF models in all cases. Figure 6 shows a comparison between the best performing FCFF and the best CNN model for turbulent channel flow Case 2. The CNN model provides an almost perfect fit to the DNS data, whereas the original FCFF model fails at some key regions. We find that both the boundary condition enforcement and the friction Reynolds number injection proposed by Fang et al. (2018) also prove successful in the case of CNN models and help the algorithm better learn the relationship between the normalized mean velocity gradient and the $u - v$ component of the normalized anisotropic Reynolds stress tensor. With a few exceptions, combining these two physics embedding techniques with the new baseline CNN model seems to be the best choice. Note that the best CNN model is able to outperform the best FCFF model using less parameters as shown in Table 3, which confirms that the new architecture is more efficient.

Next, we proceed to test the applicability of the CNN model with boundary condition enforcement and Reynolds number injection to other one-dimensional turbulent flows. Later, in Section 5.4, we will give guidelines for the reader to use interpretability to fine-tune the model to their particular one-dimensional turbulent flow of interest.

5.2 Model Results for Turbulent Couette Flow

Table 4 displays the three training-prediction cases that are examined for turbulent Couette flow. As previously mentioned in Section 2.3, the DNS data used for this flow case consists of fewer data points, for $Re_\tau = [93, 220, 500]$ the number of points is $N_y = [64, 96, 128]$, which makes it harder to learn the relationship. Although the model performance is slightly worse than that for turbulent channel flow, the R^2 scores shown in Table 5 are still satisfactory. Note that we

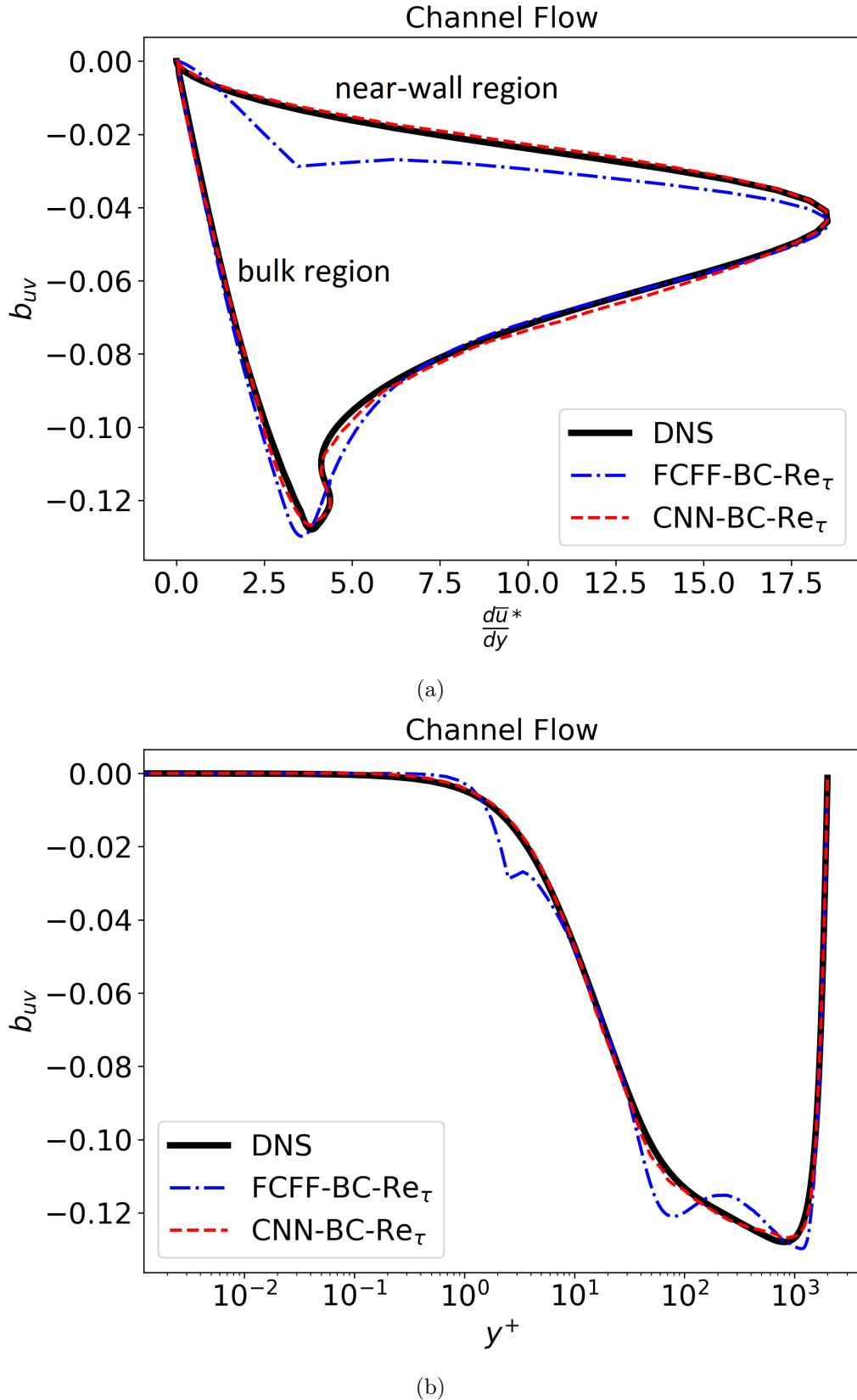


Figure 6: Comparison between FCFF-BC- Re_τ and CNN-BC- Re_τ model for turbulent channel flow Case 2 test. (a) Displays the two model predictions against the normalized mean velocity gradient profile. (b) Shows the predictions against the normalized distance from the wall in viscous units.

Table 2: R^2 score of b_{uv} predictions by various models for four turbulent channel flow training-prediction cases.

Model Type	Case 1		Case 2		Case 3		Case 4	
	Train	Test	Train	Test	Train	Test	Train	Test
CNN	0.9999	0.9299	0.9999	0.9949	0.9999	0.9978	0.9999	0.9894
CNN-BC	0.9999	0.9069	0.9999	0.9968	0.9998	0.9991	0.9998	0.9954
CNN- Re_τ	0.9999	0.9327	0.9997	0.9960	0.9996	0.9925	0.9998	0.9968
CNN-BC- Re_τ	0.9999	0.9901	0.9999	0.9982	0.9999	0.9991	0.9999	0.9953
FCFF	0.8616	0.9118	0.8688	0.8853	0.8838	0.8726	0.8981	0.8004
FCFF-BC	0.9614	0.9045	0.9645	0.9742	0.9689	0.9798	0.9825	0.9148
FCFF- Re_τ	0.8633	0.9041	0.8644	0.8839	0.8788	0.8693	0.8971	0.8011
FCFF-BC- Re_τ	0.9737	0.9473	0.9695	0.9782	0.9691	0.9814	0.9831	0.9106

Table 3: Number of trainable parameters for each model.

Model Type	Number of Trainable Parameters
CNN	10151
CNN-BC	10151
CNN- Re_τ	10166
CNN-BC- Re_τ	10166
FCFF	5351
FCFF-BC	5351
FCFF- Re_τ	10401
FCFF-BC- Re_τ	10401

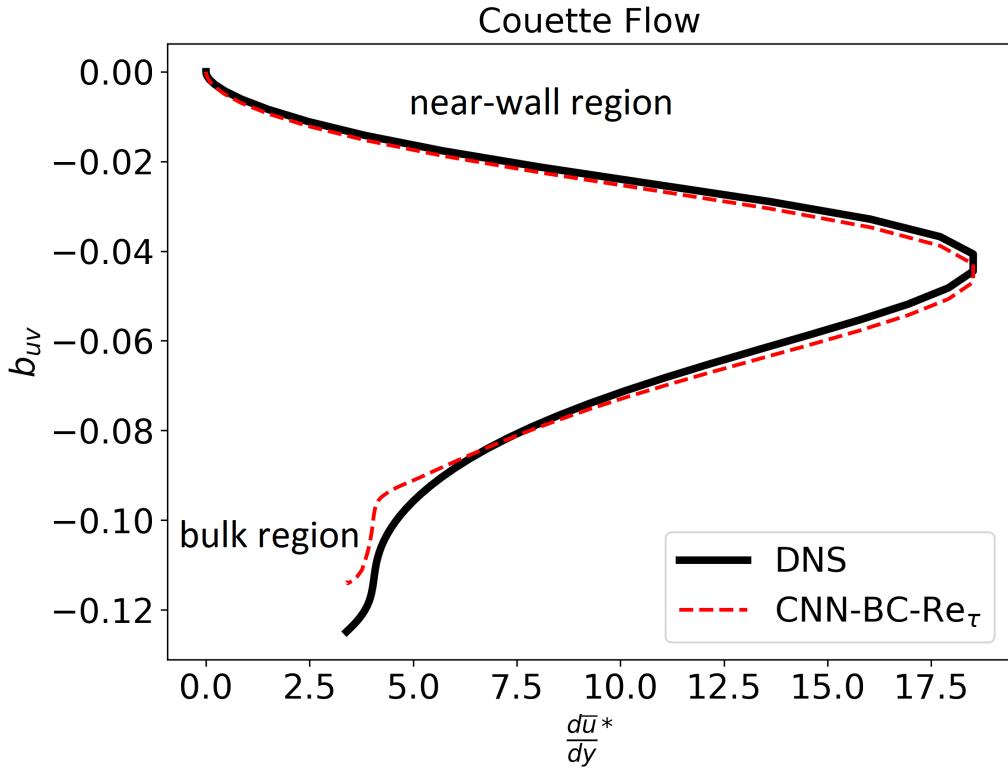
are training and testing the CNN model architecture that was optimized for turbulent channel flow on a different flow configuration. As later discussed in Section 5.4.1, fine-tuning the model for this specific case would help yield better results. Figure 7 shows the prediction for Case 2, plotted against the normalized mean velocity gradient and against the normalized distance from the wall. Overall, we find that the predictions for Couette flow are particularly good for the near-wall region, as expected thanks to the boundary condition enforcement, but the model struggles to capture the right shape as it gets closer to the bulk.

Table 4: The three training-prediction cases for turbulent Couette flow.

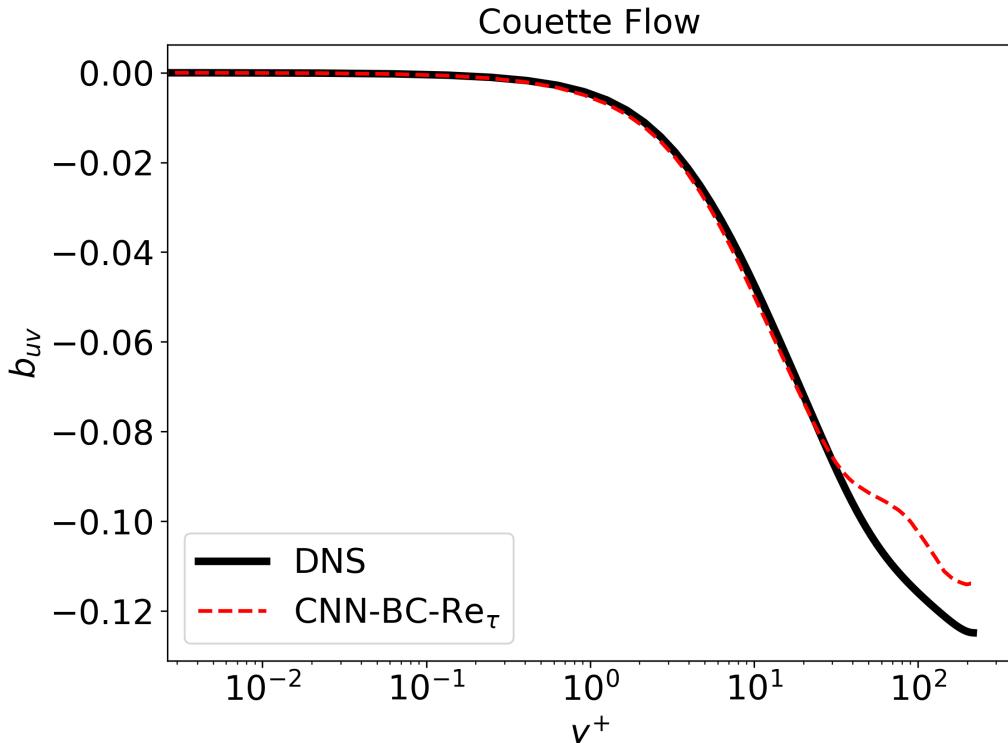
Case	Training set	Test set
1	$Re_\tau = [93, 220]$	$Re_\tau = 500$
2	$Re_\tau = [93, 500]$	$Re_\tau = 220$
3	$Re_\tau = [220, 500]$	$Re_\tau = 93$

Table 5: R^2 score of b_{uv} predictions of the best CNN model for three turbulent Couette flow training-prediction cases.

	Case 1		Case 2		Case 3	
	Train	Test	Train	Test	Train	Test
CNN-BC- Re_τ	0.9999	0.9223	0.9999	0.9575	0.9999	0.9708



(a)



(b)

Figure 7: CNN-BC- Re_τ model prediction for turbulent Couette flow Case 2 test. (a) Displays the model predictions against the normalized mean velocity gradient profile and the DNS data. (b) Shows the prediction against the normalized distance from the wall in viscous units and the DNS data.

5.3 Model Results for Turbulent Channel Flow with Wall Transpiration

As mentioned in Section 2.4, the DNS data used for turbulent channel flow with wall transpiration had three friction Reynolds numbers $Re_\tau = [250, 480, 850]$ with $N_y = [251, 385, 471]$. Table 6 shows the cases for this flow configuration and Table 7 the results. Figure 8 shows the prediction for Case 2. The introduction of a transpiration velocity substantially complicates the input-output relationship compared to the original channel flow studied by Fang et al. (2018), which forces the model to learn a more complex mapping. Nevertheless, we find that our new CNN model is able to be successfully applied to this flow and capture the features of the DNS data.

Table 6: The three training-prediction cases for turbulent channel flow with wall transpiration.

Case	Training set	Test set
1	$Re_\tau = [250, 480]$	$Re_\tau = 850$
2	$Re_\tau = [250, 850]$	$Re_\tau = 480$
3	$Re_\tau = [480, 850]$	$Re_\tau = 250$

Table 7: R^2 score of b_{uv} predictions of the best CNN model for three turbulent channel flow with wall transpiration training-prediction cases.

	Case 1		Case 2		Case 3	
	Train	Test	Train	Test	Train	Test
CNN-BC- Re_τ	0.9999	0.9865	0.9999	0.9949	0.9999	0.9459

5.4 Interpretability Results

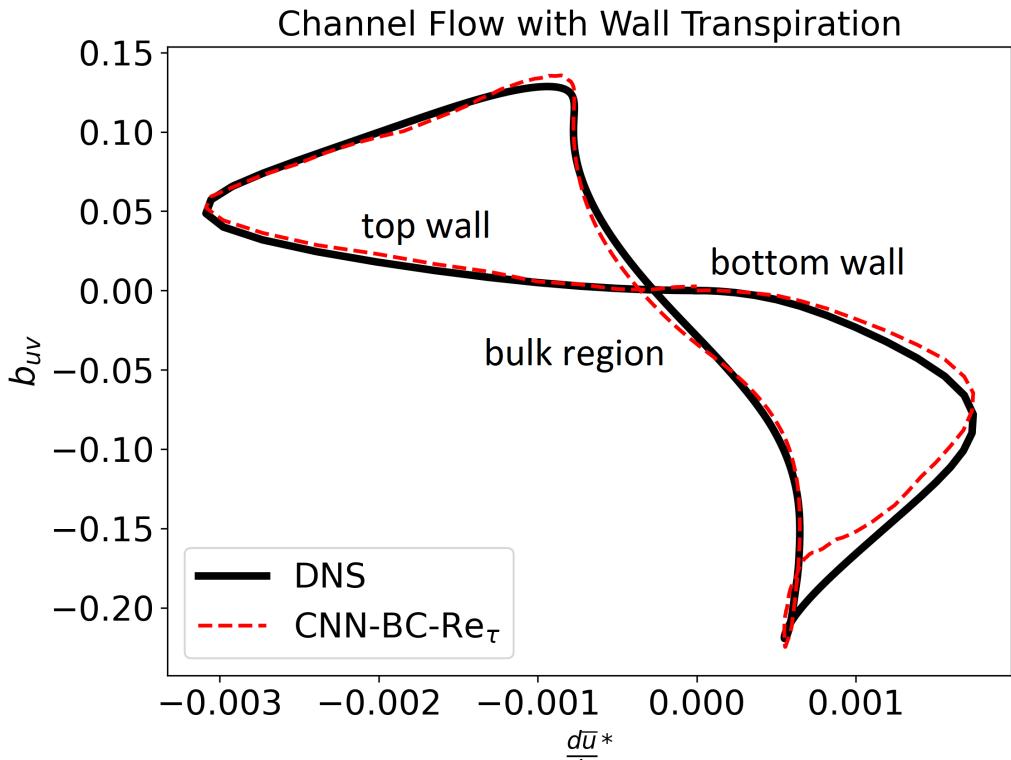
The CNN models have shown excellent performance across a range of friction Reynolds numbers as well as their applicability to different one-dimensional turbulent flows. Now, we proceed to apply the interpretability techniques described in Section 4 to the models to gain a better understanding of their decision-making process and the influence of physics embedding techniques such as boundary condition enforcement and friction Reynolds number injection.

5.4.1 Activation Visualization Results

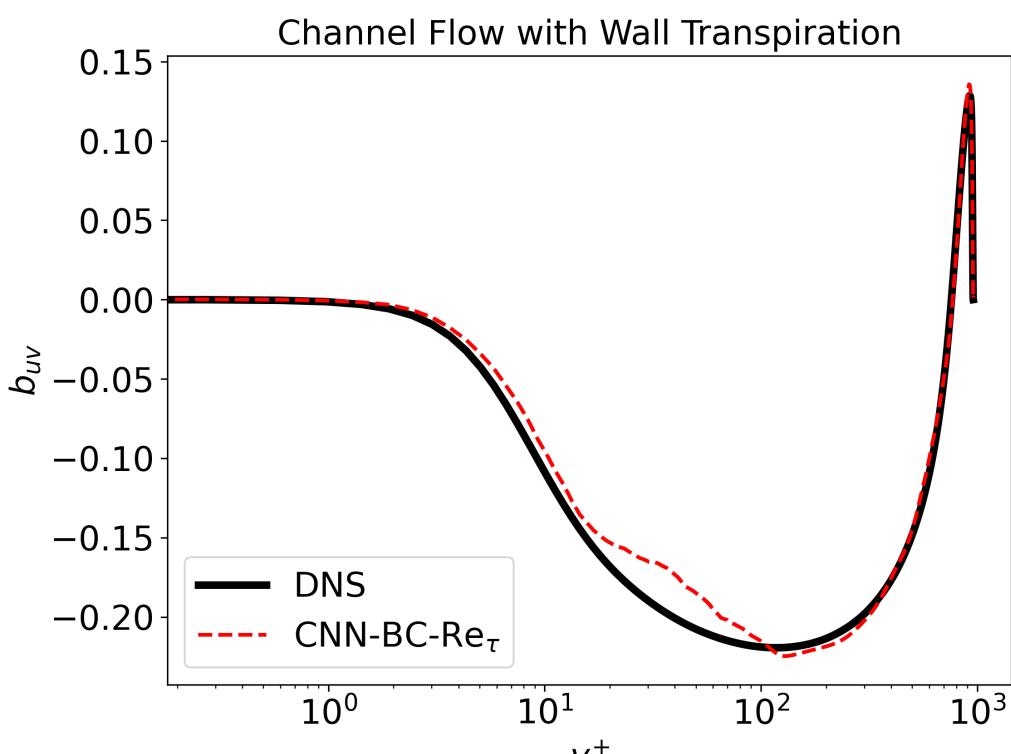
Activation visualization allows us to get an overall understanding of the transformations that the model applies to the input as well as when some filters have become redundant or more are needed to capture a complex behaviour.

For example, Figure 9 shows the activations of different layers of the CNN-BC- Re_τ model for turbulent Couette flow Case 2. Note that after the last activation, the model would apply a weighted sum operation to the output of the last convolutional layer to obtain the final prediction, and that the trainable weights for the weighted sum operation are initialized at $-\frac{1}{10}$. From the plots, we can see that one filter in layer 1 is redundant and does not get activated. A similar behavior occurs in layer 3 where two filters do not get activated. We further note that the entire last convolutional layer is also unnecessary. By removing these redundancies we can decrease the number of trainable parameters from 10166 to 4856, reduce overfitting, and improve the test R^2 error from 0.9575 to 0.9936.

As we have shown, this approach can be used to adapt and fine-tune the CNN-BC- Re_τ model originally developed for turbulent channel flow to different one-dimensional turbulent flows and avoid issues such as overfitting and underfitting. The physics of dissimilar one-dimensional



(a)



(b)

Figure 8: CNN-BC- Re_τ model prediction for turbulent channel flow with wall transpiration Case 2 test. (a) Displays the model predictions against the normalized mean velocity gradient profile and the DNS data. (b) Shows the prediction against the normalized distance from the wall in viscous units and the DNS data.

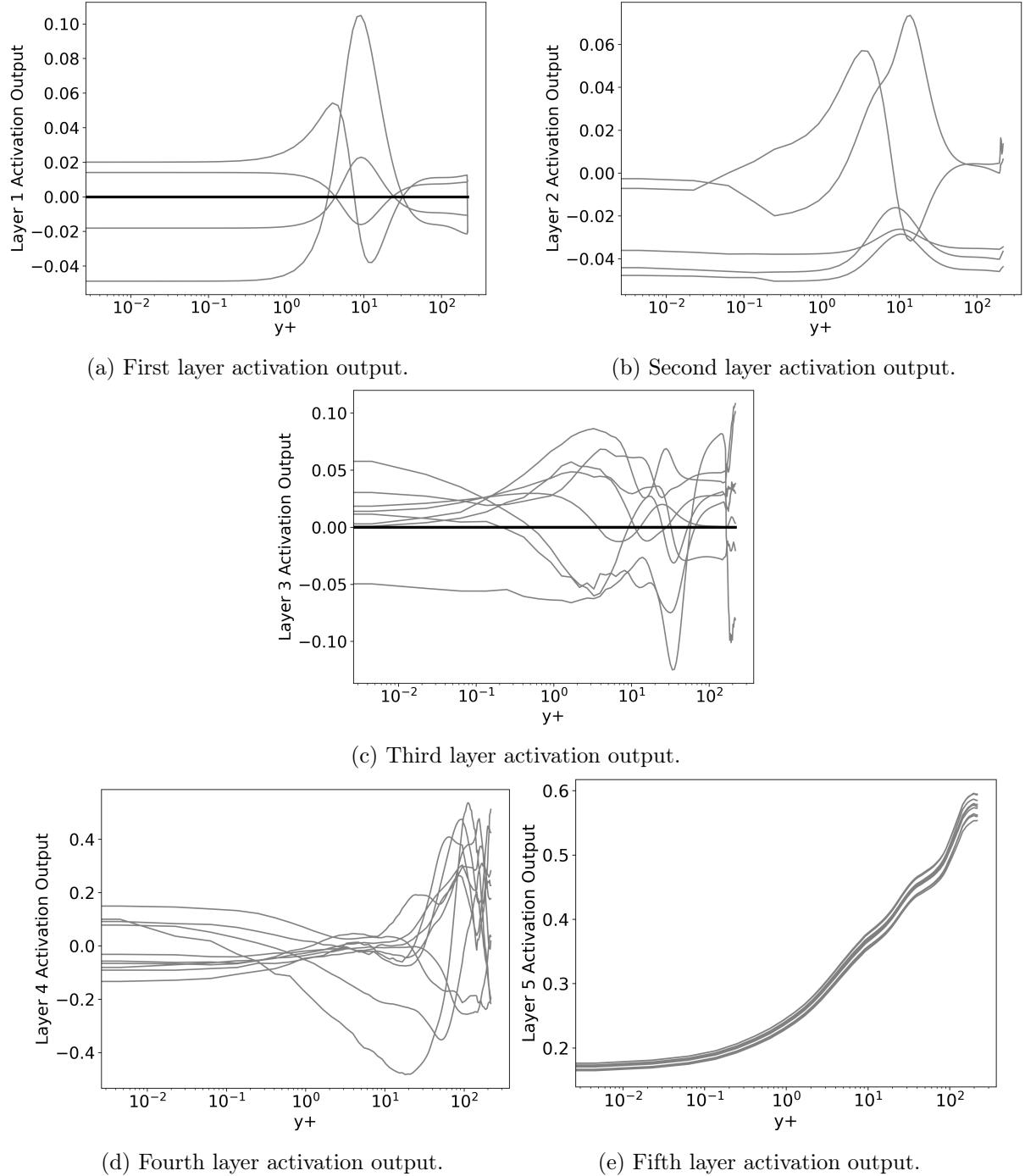


Figure 9: Activations for turbulent Couette flow Case 2 testing on $Re_\tau=220$ with the CNN-BC- Re_τ model. The curves in the plots above represent the activation outputs for each layer. The activation outputs of filters which do not get activated are highlighted with a thicker line. In (a) one filter is redundant and does not get activated. In (c) two filters are redundant and do not get activated (the activation outputs are horizontal lines). The last layer (e) is also unnecessary as all activation outputs are comparable.

turbulent flows may comprise different levels of complexity which are reflected in the input-output relationship that we are trying to capture between the mean velocity gradients and the anisotropic Reynolds stress tensor. Using this technique we can accommodate our model to several flow configurations and drive our neural network design.

5.4.2 Occlusion Sensitivity Results

We start by analysing the global occlusion sensitivity of the models. As discussed in Section 4.2, we can determine which regions of the input are important using (4.1). By this, we mean regions which, when occluded, have a greater impact on the overall model performance. To do so, we use a sliding window to systematically occlude different regions of the input. Ideally, the sliding window should be big enough to avoid capturing meaningless local noisy variations but small enough to represent key trends. Given that the number of points in the data change for every friction Reynolds number, the sliding window should be adjusted accordingly. We consistently find the near-wall region to be the most relevant, as expected. Figure 10 shows a global occlusion sensitivity plot for turbulent channel flow. The x and y axes correspond to the input and prediction of the CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 1000$. The color bar on the right denotes the important regions with light colors. As we can see from the plot, the bulk region on the left is in black, whereas the near-wall region at the top of the plot is colored in light yellow, which shows that the behaviour of the trained model aligns with the underlying physics of the problem.

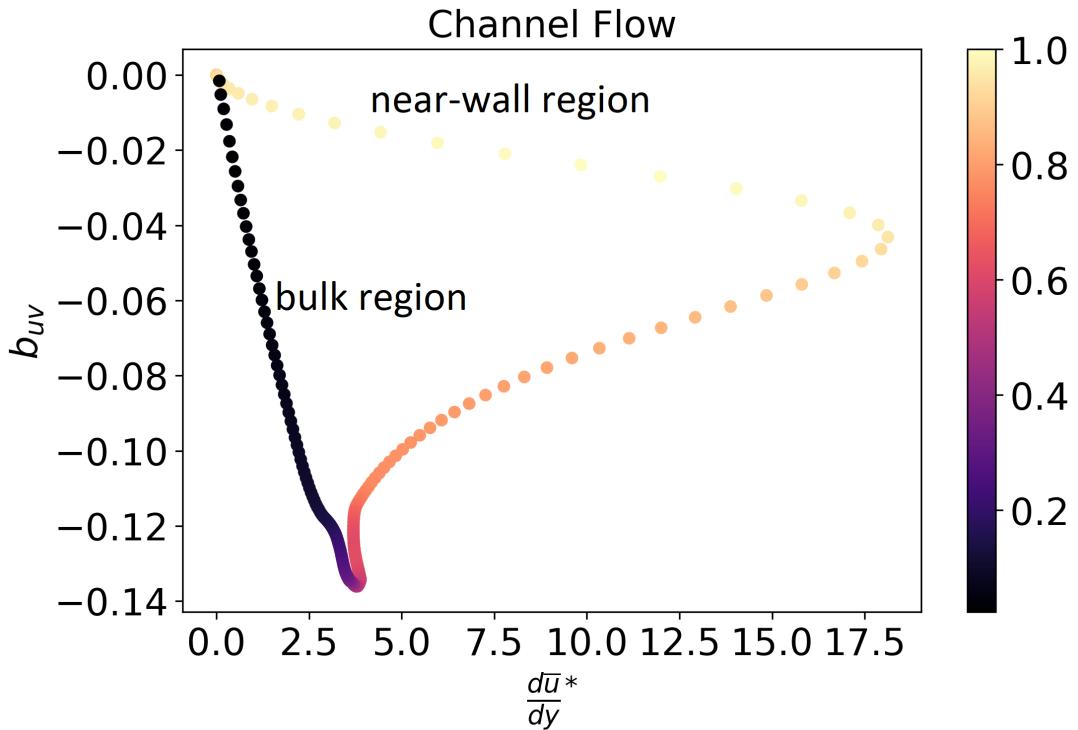


Figure 10: Global occlusion sensitivity plot for turbulent channel flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 1000$. Sliding window covers 50 entries.

Figure 11 displays the equivalent local occlusion sensitivity plot for the same model and flow configuration. In this plot, the y axis corresponds to the entry in the input normalized mean velocity gradient array and the x axis to the entry in the output prediction array, so that we can appreciate how occluding different inputs along the height of the channel affect individual b_{uv} predictions. Figure 11 shows that the model has learnt non-local relationships since the normalized velocity gradient at a given position along the channel height affects the prediction

at other locations. For example, the near-wall input entries clearly affect the last few entries of the prediction array as we can see from the yellow patch on the top right corner of the plot. On the other hand, Figure 12 shows the local occlusion sensitivity plot for turbulent channel

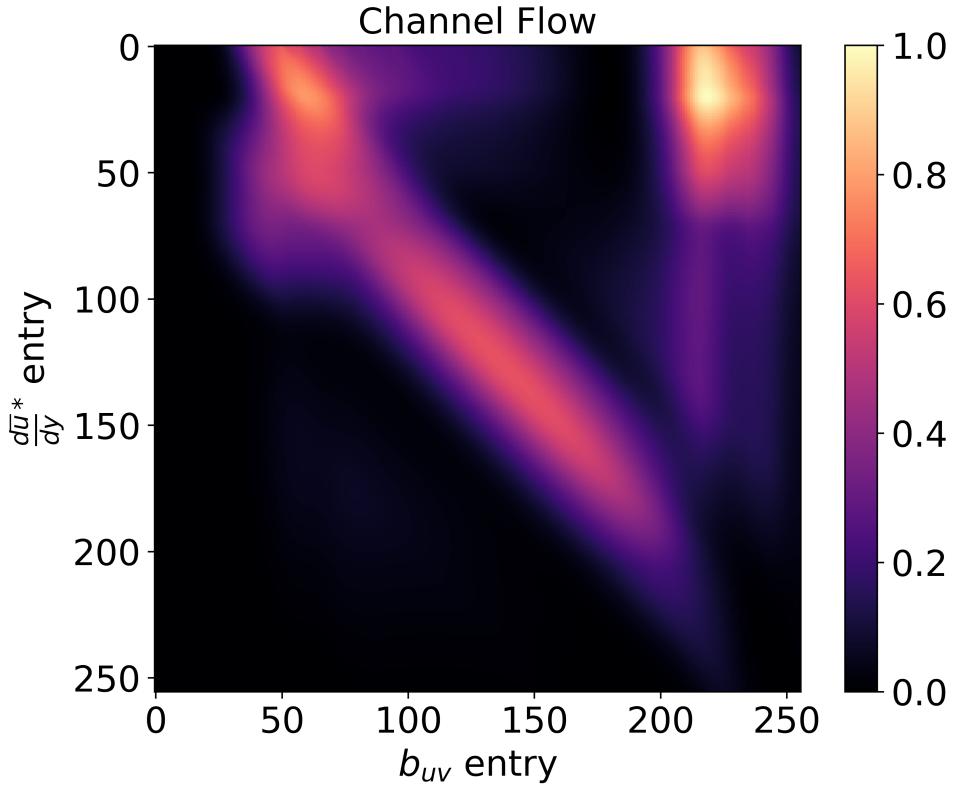


Figure 11: Local occlusion sensitivity plot for turbulent channel flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 1000$. Sliding window covers 50 entries.

flow before training the CNN-BC- Re_τ model, when the model parameters are still random. The clear differences between Figure 12 and Figure 11 corroborate that the results are not merely an artefact coming from the interpretability technique itself.

Similar global and local occlusion sensitivity plots can be obtained for turbulent Couette flow and turbulent channel flow with wall transpiration. Figure 13 and Figure 14 display sensitivity plots for the CNN-BC- Re_τ model applied to Couette Flow Case 3. The model also pays attention to the near-wall region in this case, as seen in Figure 13. According to Figure 14, changes in the first entries of the input array seem to especially affect the last few entries of the output. We indeed found that although the model was obtaining good results near the wall, its predictions for the bulk region tended to fluctuate for similar training losses. We can conclude that the plot reflects that changes in the near wall region input substantially affect the bulk region prediction. Lastly, Figure 15 and Figure 16 represent the occlusion sensitivity of the model for channel flow with wall transpiration. We can see that the model focuses on the near-wall regions where most of the physics takes place and it ignores the bulk, which corresponds to the straight black line near the center of the plot in Figure 15.

5.4.3 Gradient-based Sensitivity Results

We aim at measuring the relative importance of each entry in the input array on the final model prediction by looking at the magnitude of the gradients with respect to the loss as previously described in Section 4.3. In this context, bigger gradients mean greater importance. Based on (4.3) we can plot smooth global gradient-based saliency maps. Figure 17 displays a plot for

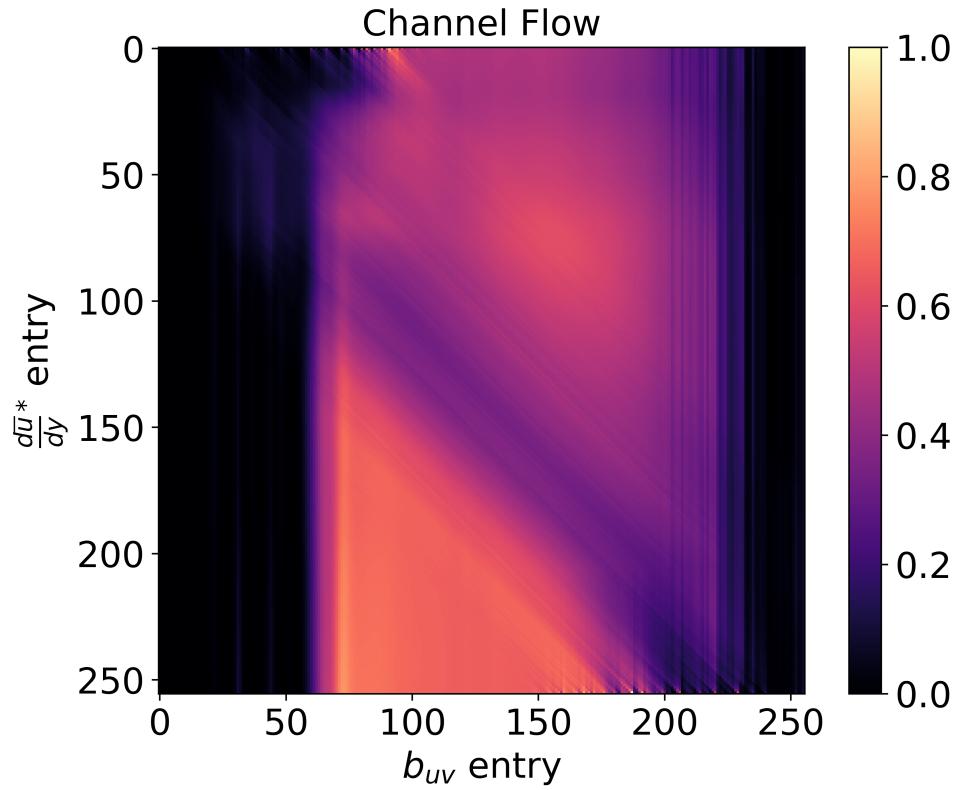


Figure 12: Local occlusion sensitivity plot for turbulent channel flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 1000$ before training.

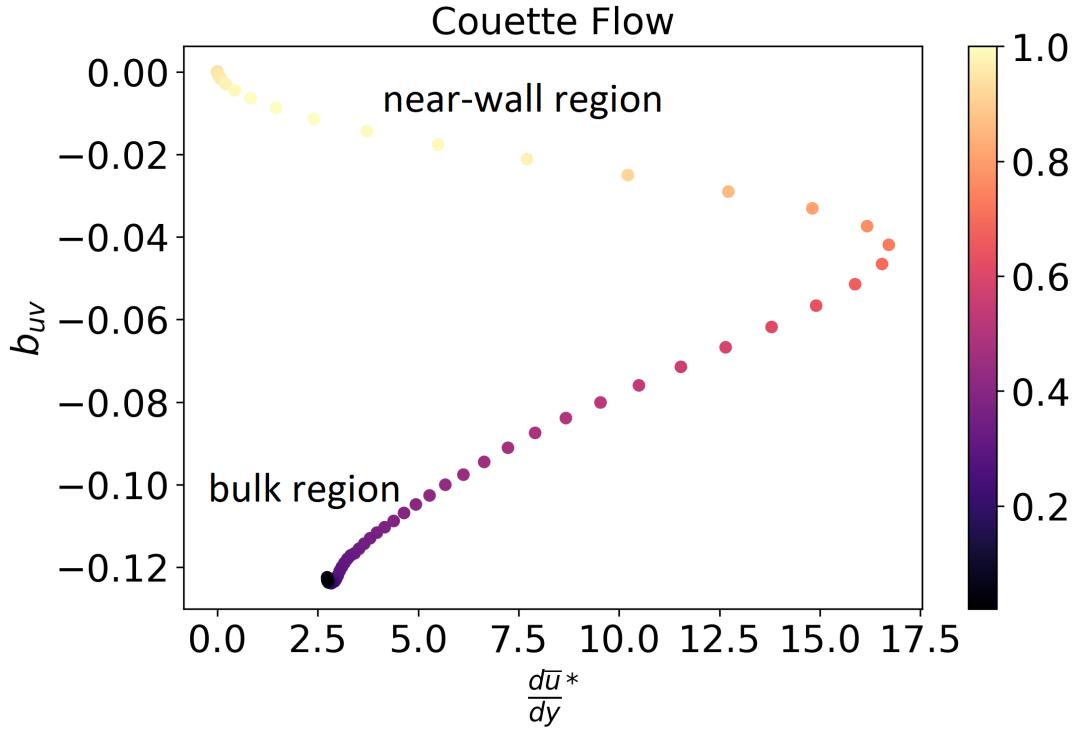


Figure 13: Global occlusion sensitivity plot for turbulent Couette flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 93$. Sliding window covers 50 entries.

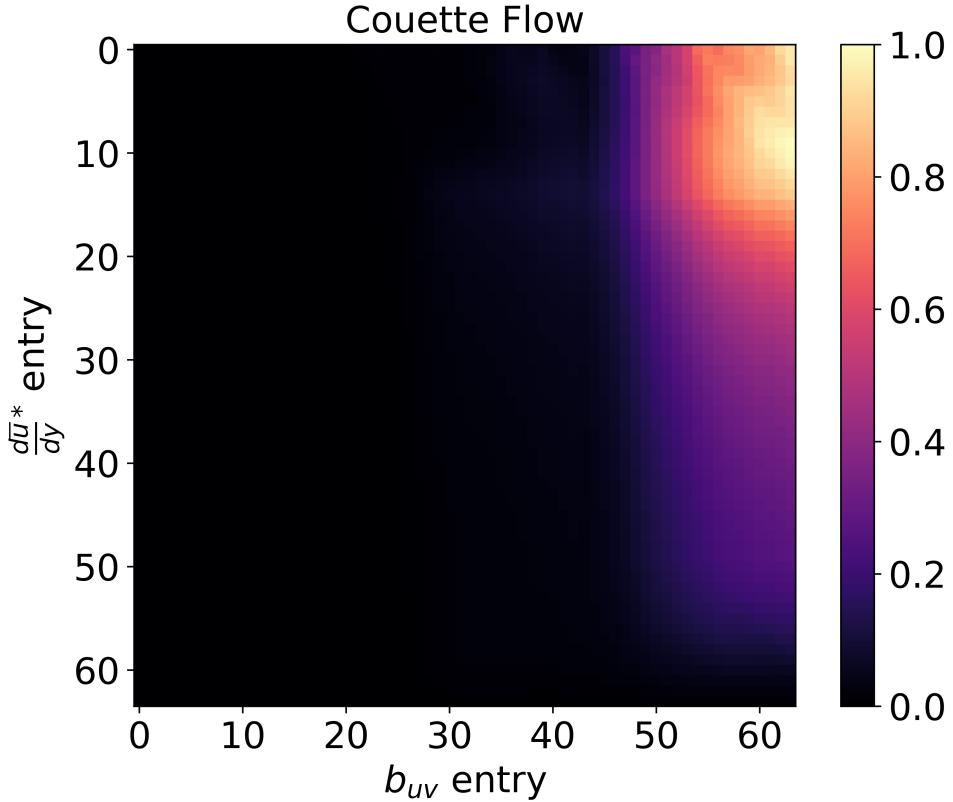


Figure 14: Local occlusion sensitivity plot for turbulent Couette flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 93$. Sliding window covers 50 entries.

the CNN-BC- Re_τ model for turbulent channel flow Case 3 testing on friction Reynolds number $Re_\tau = 1000$. As in Figure 10, Figure 17 also highlights the near-wall region. It does, however, slightly highlight other regions. This could be attributed to artifacts coming from the batch normalization layers within the network, which only partially estimate the mean and variance on individual batches and therefore introduce noise. Overall, we find gradient-based and occlusion sensitivity maps tend to agree, but the latter seems to be more consistent and reliable.

6 Conclusion

The RANS equations are widely used in turbulence modeling applications but accurately modeling the Reynolds stress tensor remains a challenging problem. Recently, researchers have started to explore machine learning applications to turbulence modelling using novel deep neural network architectures trained with DNS data. However, little emphasis has been given to the interpretability of the models. Applying human expertise to tune and improve machine learning models is necessary, especially in such a complex field as turbulence. Understanding the algorithm decision-making process, and how different physics embedding approaches influence it, is key to improving the models.

In this work, we focused on developing better performing neural networks for turbulent one-dimensional flows and applied several interpretability techniques to recognize which inputs have the greatest effect on the accuracy of the model prediction. We compared the results obtained with our CNN models for turbulent channel flow to those using the original FCFF model by Fang et al. (2018) and obtained considerable improvements and a good fit to the data. Next, we demonstrated the applicability of our model to other one-dimensional flows, for which it also proved successful. Finally, we proposed a number of interpretability techniques which were used

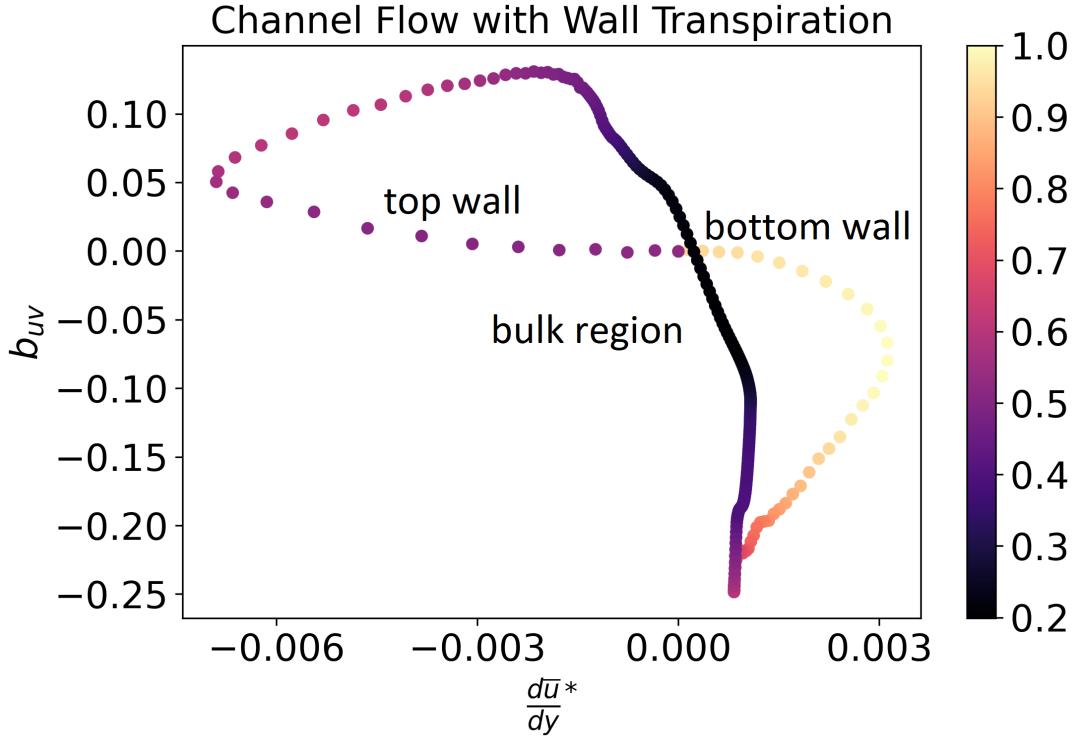


Figure 15: Global occlusion sensitivity plot for turbulent channel flow with wall transpiration, CNN-BC- Re_τ model for Case 3 testing on $Re_\tau = 250$. Sliding window covers 50 entries.

as a guideline to fine-tune the model to different flow configurations and to check that the model behaviour was indeed in line with our understanding of the underlying physics of the problem.

There are several directions for future work, such as testing the models on more turbulent one-dimensional flow configurations and on higher dimensional turbulent flows. The latter will likely require developing new physics embedding techniques as well as more sophisticated interpretability approaches. Furthermore, in some exploratory studies outside the scope of the present work, we also found that applying transfer learning can substantially speed up the training process of the CNN models discussed in this paper. Starting from the model weights for channel flow, the number of epochs required to train the model for Couette flow and channel flow with wall transpiration can be reduced by one order of magnitude compared to retraining the model from scratch. Transfer learning could be explored in future work to help improve the training time of neural networks applied to turbulence modelling.

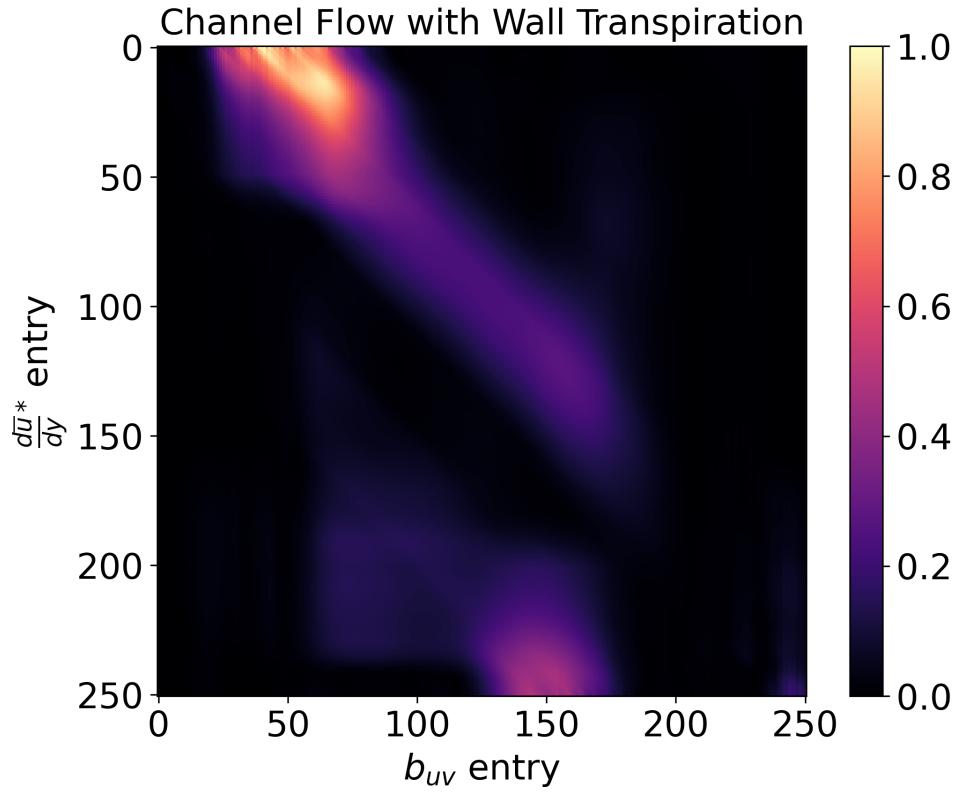


Figure 16: Local occlusion sensitivity plot for turbulent channel flow with wall transpiration, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 250$. Sliding window covers 50 entries.

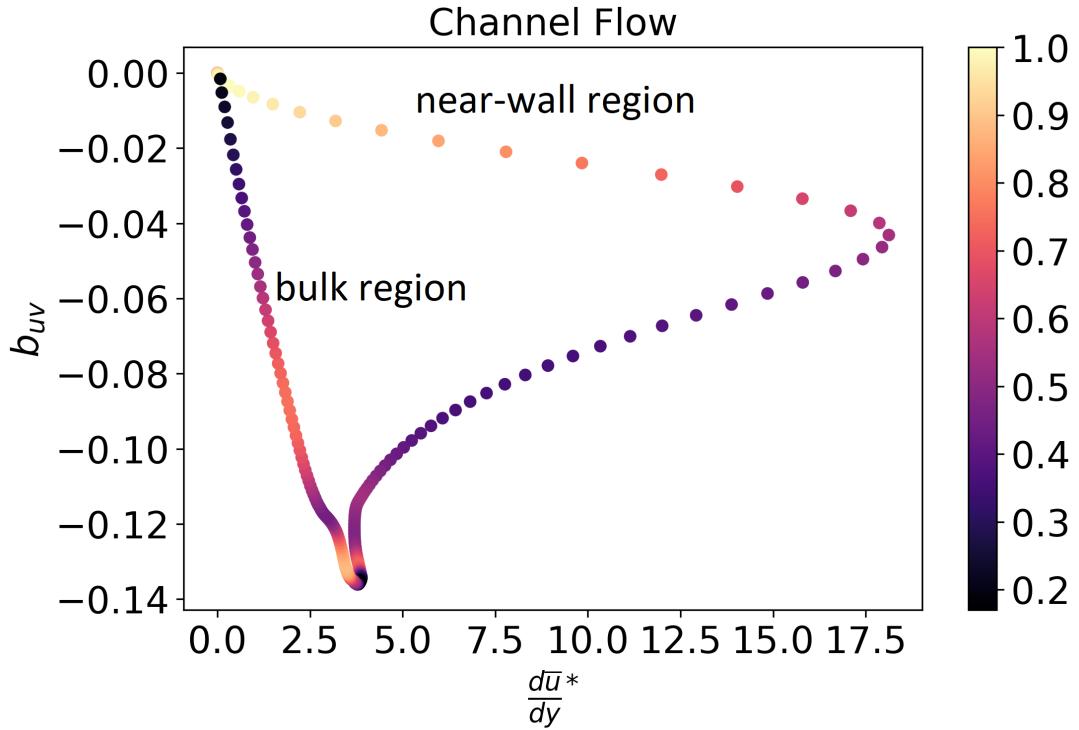


Figure 17: Global smooth gradient-based sensitivity plot for turbulent channel flow, CNN-BC- Re_τ model for Case 3 testing on friction Reynolds number $Re_\tau = 1000$.

References

- Ancona, M., Ceolini, E., Öztireli, A. C., and Gross, M. (2017). A unified view of gradient-based attribution methods for deep neural networks. *ArXiv*, abs/1711.06104.
- Avsarkisov, V., Oberlack, M., and Hoyas, S. (2014). New scaling laws for turbulent poiseuille flow with wall transpiration. *Journal of Fluid Mechanics*, 746:99–122.
- Barkley, D. and Tuckerman, L. (2007). Mean flow of turbulent–laminar patterns in plane couette flow. *Journal of Fluid Mechanics*, 576:109–137.
- Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *SIAM Review*, 60.
- Carvalho, D. V., Pereira, E. M., and Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8).
- Chakraborty, S., Tomsett, R., Raghavendra, R., Harborne, D., Alzantot, M., Cerutti, F., Srivastava, M., Preece, A., Julier, S., Rao, R. M., Kelley, T. D., Braines, D., Sensoy, M., Willis, C. J., and Gurram, P. (2017). Interpretability of deep learning models: A survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6.
- Chen, H., Kandasamy, S., Orszag, S., Shock, R., Succi, S., and Yakhot, V. (2003). Extended boltzmann kinetic equation for turbulent flows. *Science*, 301(5633):633–636.
- Chollet, F. (2018). *Deep learning with Python*. Manning, Shelter Island, NY.
- Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning.
- Duraisamy, K., Iaccarino, G., and Xiao, H. (2019). Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51(1):357–377.
- Fang, R., Sondak, D., Protopapas, P., and Succi, S. (2018). Deep learning for turbulent channel flow.
- Gatski, T. (2004). Constitutive equations for turbulent flows. *Theoretical and Computational Fluid Dynamics*, 18(5):345–369.
- Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2019). Explaining explanations: An overview of interpretability of machine learning.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- Hoyas, S. and Jiménez, J. (2006). Scaling of the velocity fluctuations in turbulent channels up to $Re = 2003$. *Physics of Fluids - PHYS FLUIDS*, 18.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

- Johansson, A. (2002). Engineering turbulence models and their development, with emphasis on explicit algebraic reynolds stress models. In *Theories of Turbulence*, pages 253–300. Springer.
- Kaandorp, M. (2018). Machine learning for data-driven rans turbulence modelling.
- Kaandorp, M. L. and Dwight, R. P. (2020). Data-driven modelling of the reynolds stress tensor using random forests with invariance. *Computers & Fluids*, 202:104497.
- Kiefer, J. and Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466.
- Kim, B., Khanna, R., and Koyejo, O. O. (2016). Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in neural information processing systems*, pages 2280–2288.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Lee, M. and Moser, R. (2018). Extreme-scale motions in turbulent plane couette flows. *Journal of Fluid Mechanics*, 842:128–145.
- Lee, M. and Moser, R. D. (2015). Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415.
- Ling, J., Jones, R., and Templeton, J. (2016a). Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 318.
- Ling, J., Kurzawski, A., and Templeton, J. (2016b). Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166.
- Lozano-Durán, A. and Jiménez, J. (2014). Effect of the computational domain on direct simulations of turbulent channels up to $Re_\tau = 4200$. *Physics of Fluids (1994-present)*, 26:–.
- Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *ArXiv*, abs/1706.07269.
- Millstein, F. (2018). *Convolutional Neural Networks in Python: Beginner's Guide to Convolutional Neural Networks in Python*. CreateSpace Independent Publishing Platform.
- Molnar, C. (2019). *Interpretable Machine Learning*. lulu.com. <https://christophm.github.io/interpretable-ml-book/>.
- Pope, S. B. (1975). A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(2):331–340.
- Prechelt, L. (1998). Early stopping-but when? *Neural Networks: Tricks of the trade*, pages 553–553.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps.
- Smilkov, D., Thorat, N., Kim, B., Viégas, F., and Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise.

- Song, X., Zhang, Z., Wang, Y., Ye, S., and Huang, C. (2019). Reconstruction of rans model and cross-validation of flow field based on tensor basis neural network.
- Speziale, C. G. (1991). Analytical methods for the development of reynolds-stress closures in turbulence. *Annual Review of Fluid Mechanics*, 23(1):107–157.
- Tracey, B. D., Duraisamy, K., and Alonso, J. (2015). A machine learning strategy to assist turbulence model development.
- Wang, J.-X., Wu, J.-L., and Xiao, H. (2017). Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Phys. Rev. Fluids*, 2:034603.
- Wu, J.-L., Xiao, H., and Paterson, E. (2018). Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Phys. Rev. Fluids*, 3:074602.
- Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks.
- Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K. (2021). A survey on neural network interpretability.
- Zhang, Z. and Duraisamy, K. (2015). Machine learning methods for data-driven turbulence modeling.
- Zhu, Y. and Dinh, N. (2020). A data-driven approach for turbulence modeling.