MULTIFIDELITY DATA FUSION IN CONVOLUTIONAL ENCODER/DECODER NETWORKS

A PREPRINT

Lauren Partin[1], Gianluca Geraci[2], Ahmad Rushdi[3], Michael S. Eldred[2], and Daniele E. Schiavazzi[1]

[1]Department of Applied and Computational Mathematics and Statistics
University of Notre Dame, Notre Dame, IN, USA
[2]Center for Computing Research, Optimization and UQ Group
Sandia National Laboratories, Albuquerque, NM, USA
[3]Institute for Human-Centered Artificial Intelligence
Stanford University, Stanford, CA, USA

May 10, 2022

ABSTRACT

We analyze the regression accuracy of convolutional neural networks assembled from encoders, decoders and skip connections and trained with multifidelity data. Besides requiring significantly less trainable parameters than equivalent fully connected networks, encoder, decoder, encoder-decoder or decoder-encoder architectures can learn the mapping between inputs to outputs of arbitrary dimensionality. We demonstrate their accuracy when trained on a few high-fidelity and many low-fidelity data generated from models ranging from one-dimensional functions to Poisson equation solvers in two-dimensions. We finally discuss a number of implementation choices that improve the reliability of the uncertainty estimates generated by Monte Carlo DropBlocks, and compare uncertainty estimates among low-, high- and multifidelity approaches.

1 Introduction

Analyzing physical phenomena through their mathematical or numerical modeling is a common practice in engineering and science, providing the analyst with the ability to predict the behavior of a system outside of a limited number of observations. Simulation of complex phenomena, for example characterized by multiple interacting physics, may require a substantial computational effort, and the availability of sufficient resources may be a key factor in the ability to answer the scientific questions of interest. However, it is often possible to combine accurate but expensive high-fidelity simulations with lower-fidelity simulations that provide approximations at a reduced cost, in order to optimize efficiency while retaining accuracy.

This study focuses on generating multifidelity (MF) surrogate models designed to combine information from a few high-fidelity (HF) model solutions and many low-fidelity (LF) approximations of varying accuracy. More specifically, we focus on data-driven multifidelity surrogates in the machine learning context. Previous work considered student-teacher networks with the ability to handle datasets with variable annotation quality [1], surrogates trained using transfer learning between two model fidelities [2], and fully connected neural networks combining three sub-networks designed to learn a LF representation, the correlation between a LF and a HF representation, and to minimize a physics-based residual loss [3]. Other approaches utilize Bayesian neural networks [4], or combine convolutional and fully connected networks to learn the discrepancy between increasingly fine discretizations of a given PDE solution, projected on a common mesh [5].

Our approach is inspired by the recent successes in image classification and segmentation tasks shown by deep convolutional encoder-decoder networks (see, e.g., [6, 7]). While multifidelity data fusion has been mainly demonstrated for fully connected networks or for ensembles of hybrid convolutional and fully connected networks [5], no approach has focused on convolutional networks assembled from encoders, decoders and skip connections, where the model fideli-

ties are learned simultaneously, following an *all-at-once* training paradigm. Convolutions are essential to reduce the number of weights with respect to fully connected networks when the input, the output or both are high-dimensional, as discussed in our recent work [8, 9].

We also focus on quantifying the *predictive uncertainty* in the network outputs, i.e., we want to characterize the variability of the predicted quantities of interest, a paradigm commonly referred to as "UQ for ML", analyzing the uncertainty in predictions that are inherent when using a machine-learned surrogate model. We consider this as a *model form* uncertainty that relates to how the information flows through the selected multifidelity network, as opposed to other paradigms where a deterministic machine learning model is employed as an inexpensive surrogate for uncertainty quantification studies (referred to as "ML for UQ").

Many different approaches have been proposed in the literature to quantify predictive uncertainty in neural network outputs [10, 11, 12]. Among these, dropout layers [13] offer a simple and computationally appealing solution to drop neurons at random, providing, at the same time, regularization and variance estimates. Their interpretation in terms of an ensemble of network architectures has also been investigated in the literature [14, 15]. However, their performance has been mainly assessed on neural networks with fully connected layers. In this study, we use DropBlocks [16], i.e., adaptations of dropout layers showing improved performance on convolutional architectures. This study extends previous results from our research group in two directions. In [8, 9], we focused on two separate questions, i.e., the problem of identifying network hyperparameters leading to optimal accuracy, and the problem of understanding the effect of hyperparameter selection on the variability of network predictions. Here we unify these two perspectives by studying networks providing the best trade off between accuracy and uncertainty. In addition, we show new results for the characterization of uncertainty in the one-dimensional and low- to high-dimensional test cases.

This paper is organized as follows: Section 2 introduces the problems of interest including one-dimensional function approximation from MF datasets, and prediction of high-dimensional responses from computational fluid dynamics solvers. Section 3 introduces the convolutional network architectures used in the study. Uncertainty estimates through Monte Carlo DropBlock are discussed in Section 3.4. Results are summarized in Section 4, while conclusions and future work are finally discussed in Section 5. For the interested reader, implementation details are reported in the appendix.

2 Problem description

We study the approximation performance of multifidelity networks on three different problem instances. We begin with function approximation, where we show how a convolutional architecture can be designed to accurately learn a map between inputs and outputs, even in a single dimension. We then consider *dense regression* problems where inputs and outputs are images of the same size (i.e., having equal dimensionality). Finally, we present a multifidelity architecture for *low- to high-dimensional regression*, where a high-dimensional output is predicted from a low-dimensional input. For all these cases, we examine how low-fidelity representations can be leveraged to accelerate training and improve the accuracy of high-fidelity predictions from limited data.

2.1 One-dimensional multifidelity function approximation

We consider two examples, each consisting of two correlated LF and HF functions, with very few available HF and relatively more LF training examples [3]. The first example consists of two linearly correlated continuous functions, defined as

$$y_L(x) = (1/2)(6x-2)^2\sin(12x-4) + 10(x-1/2) - 5 \quad (1)$$

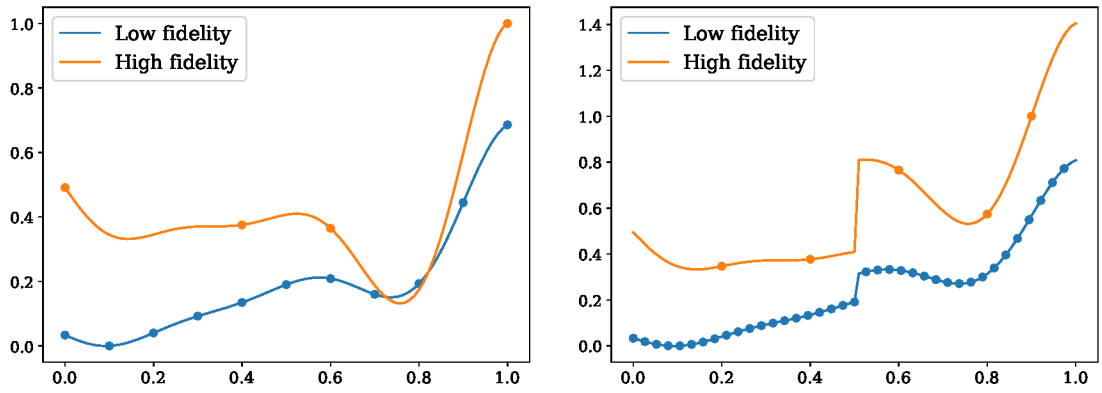
$$y_H(x) = (6x-2)^2\sin(12x-4), \quad (2)$$

where 11 and 4 samples, are provided for y_L and y_H, respectively, as shown in Fig. 1(a). In the second example, we consider two linearly correlated discontinuous functions expressed as

$$y_L(x) = \begin{cases} l(x) = 0.5(6x-2)^2\sin(12x-4) + 10(x-0.5) - 5 & 0 \le x \le 0.5 \\ 3 + l(x) & 0.5 < x \le 1 \end{cases} \quad (3)$$

$$y_H(x) = \begin{cases} h(x) = 2y_L(x) - 20x + 20 & 0 \le x \le 0.5 \\ 4 + h(x) & 0.5 < x \le 1, \end{cases} \quad (4)$$

with 38 and 5 training samples for y_L and y_H, respectively, as shown in Fig. 1(b). We only consider linear correlations in these examples, since these suffice to show the performance of the proposed network for one-dimensional problems and provide a point of comparison with existing literature [3]. However, this is not explored further, since fully-connected and convolutional architectures contain a similar number of weights for one-dimensional regression problems, and therefore convolutional networks provide no practical computational advantage.



(a) (b)

Figure 1: LF and HF functions and respective training locations from (a) Eqs. (1), (2) and (b) Eqs. (3), (4). Function values are rescaled such that $y_H(x), y_L(x) \in [0, 1]$ for x in the training set.

2.2 Dense regression

For the high-dimensional dense regression case, we focus on an application in computational fluid dynamics, where we are interested in predicting the pressure distribution in a fluid domain Ω_f from a noisy binary mask and its velocities. The binary mask identifies the fluid region and is referred to as the scalar *concentrations*. The pressure, up to a constant, can be computed through a reformulation of the incompressible Navier Stokes equations as a Poisson pressure equation with appropriate boundary conditions [17]. This approach, however, may require the solution of a partial differential equation on a large computational grid and training a neural network could provide a much faster and computationally attractive alternative. Note that, unlike physics-informed neural networks (PINN [18]), we would like to learn a relation between high-dimensional concentration/velocity inputs and high-dimensional pressure outputs (unlike the one-dimensional problem discussed in Section 2.1), rather than pressure as a function of space and time.

Under the assumption that no body force is acting on the fluid, the Poisson equation for the pressure p can be written as

$$\Delta p = \boldsymbol{\nabla} \cdot \mathbf{f} = \boldsymbol{\nabla} \cdot \left[-\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) + \mu \, \Delta \mathbf{u} \right], \tag{5}$$

and only Neumann boundary conditions are applied equal to the flux of $\mathbf{f}$ across the smooth boundary $\partial\Omega_f$ of Ω_f [17].

The problem is discretized through a structured grid Ω with $m_x \cdot m_y \cdot m_z = m$ cells, where the cells belonging to the fluid region are identified by components equal to 1 in a binary *concentration* array $c_b \in \{0, 1\}^m$. However, in practice, noise makes the concentration non-binary, with *measured* concentration expressed as $c \in \mathbb{R}^m$. Our objective is to train a neural network surrogate to quickly evaluate the map

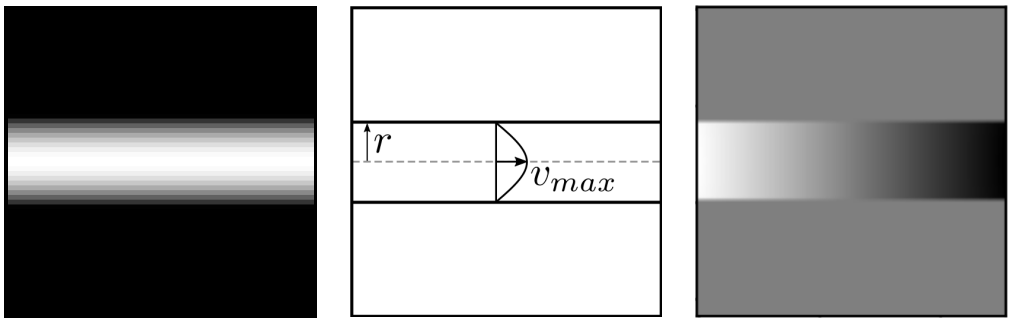
$$f(\mathbf{c}, \mathbf{u}) = \mathbf{p}, \text{ where } f : \mathbb{R}^m \times \mathbb{R}^{m \times 3} \to \mathbb{R}^m, \tag{6}$$

which, given the noisy concentrations and velocity distributions over Ω, returns the spatial pressure distribution on Ω_f. Our main goal in this paper is to investigate the possibility to increase training efficiency and accuracy using a multifidelity dataset containing pressure representations with increasingly coarser resolution. A multifidelity training set is obtained by combining a small number of HF examples, resulting from a Poisson pressure equation finite element solver, with a large number of solutions from the same solver, but evaluated on coarser discretizations. Although the parabolic velocities and linear relative pressures associated with Poiseuille flow represent a rather smooth field to be emulated by the network, and therefore may appear to offer a rather simplistic test case, we note how the change in the flow domain introduces a fair amount of complexity. This study therefore focuses on the development of highly accurate U-Net-like convolutional architectures for this flow, and provides an initial exploration of the method's robustness to noise and bias in the low-fidelity approximants. Future work will focus on quantifying the performance of the proposed approach on more complex flow configurations.

2.3 Low- to high-dimensional regression

We consider a problem with the same pressure outputs as that in Section 2.2, but with only two inputs, i.e., the radius r of the cylindrical fluid domain and the maximum velocity v_{max}, as these parameters are sufficient to fully specify

the geometry and velocity distribution in a Hagen-Poiseuille flow, as shown in Fig. 2(b). In other words, this case represents the surrogate construction for a random field (the pressure) given only two input parameters. This low-to-high dimensional regression problem may not be easy to solve for traditional surrogate-based approaches. For example, techniques like generalized polynomial chaos (gPC [19]) could be used to obtain scalar pressure estimates at each pixel in the fluid domain, but additional structure (e.g. *modes* informed from dimensionality reduction algorithms) would need to be specified to account for the spatial correlation of the resulting pressure field.



(a) (b) (c)

Figure 2: Example of Poiseuille flow. Velocity profile (a). Test case parameterization in terms of the fluid region radius and maximum velocity (b). Pressure result (c).

3 Network architectures

For all problems discussed above, we employ a network architecture assembled from convolutional encoders, decoders and skip connections. A convolutional encoder [20] is composed of alternating layers of convolutions and pooling (i.e., downsampling), generating a compressed feature representation. A convolutional decoder, on the other hand, is composed of alternating layers of convolutions and upsampling. Skip connections are finally added to mitigate the loss of information due to downsampling, and counteract the vanishing gradient problem (see, e.g. [21]). For dense regression problems, the encoder and decoder are symmetric, and padding is applied so that the number of pixels in the network input and output is the same.

In the next sections, we describe the three specific network architectures used to address the problems presented in Section 2.1, Section 2.2 and Section 2.3, respectively. Section 3.4 provides an overview of DropBlock layers and their use in uncertainty quantification. In addition, Section 3.5 discusses how the information from low- and high-fidelity models is assembled in a network.

3.1 Decoder-encoder architecture for one-dimensional regression

For one-dimensional regression (Section 2.1), a single LF predictor (as opposed to multiple LF predictors for the networks discussed in the next sections) is generated by the network. Denote z as the concatenation between the LF predictor and its x coordinate. The HF predictor is obtained by summing two contributions

1. a convolution applied to z, designed to capture the linear correlation between the HF and LF outputs; and
2. the convolution of z in two layers which are separated by a nonlinear activation, designed to capture the nonlinear correlation between the HF and LF outputs.

Additive skip connections are used to facilitate the exchange of information between the decoder and the encoder. The network layout is illustrated in Fig. 3. Additional details are included, for the interested reader, in the appendix.

3.2 Encoder-decoder architecture for dense regression

This architecture resembles the popular U-Net [22] that has shown remarkable performance in terms of accuracy and training speed for segmentation tasks, even under limited training data [23]. We consider both input and output images with 64×64 pixels, where the input is characterized by three channels (one concentration and two velocity components) and the output by a single channel (the pressure). A simple identity replaces the ReLU activation after the final convolution layer.

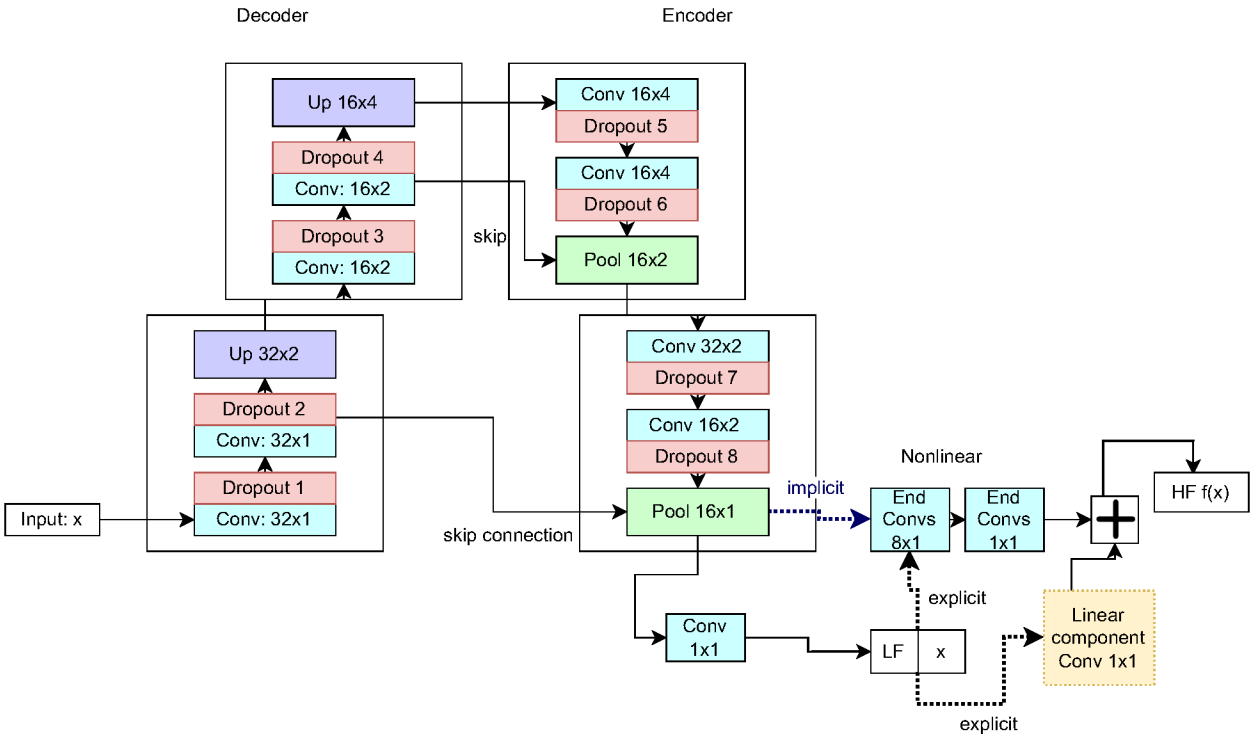


Figure 3: Multifidelity decoder-encoder convolutional network architecture for one-dimensional regression. The HF output is a linear combination of the nonlinear and linear portion of the network, $\text{HF} = \gamma \cdot \text{Linear}(\text{LF}, x) + (1 - \gamma) \cdot \text{Nonlinear}(\text{LF}, x)$, where γ is a parameter learned during training.

A multifidelity network is obtained by extracting a LF representation of increasing resolution at each stage of the decoder. A term for each LF predictor is then added to the loss function, so these LF representations are accurately learned. This is depicted in Fig. 4 where the models are ordered as LF1, LF2, LF3, HF, i.e., from the coarsest to the finest resolutions. For the interested reader, additional details on this network are included in the appendix.

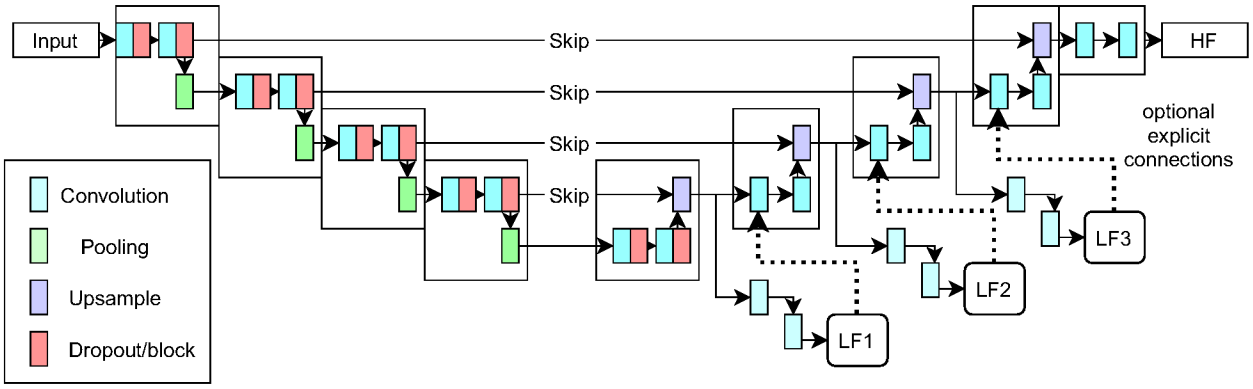


Figure 4: Multifidelity decoder-encoder convolutional network architecture for high-dimensional dense regression.

3.3 Decoder architecture for low- to high-dimensional regression

The network selected for low- to high-dimensional regression is shown in Fig. 5. The two dimensional input is upsampled to generate 64×64 output images. This is achieved with a single decoder, choosing the network depth and padding (see appendix) to enforce the correct output dimensions, i.e. 6 upsampling layers, each with a scale factor of 2.

Similar to the network in the previous section, a LF prediction is generated at each decoder stage, ordered as LF1, LF2, LF3, HF, i.e., from the coarsest to the finest resolution, respectively. For the interested reader, additional details on this network are included in the appendix.

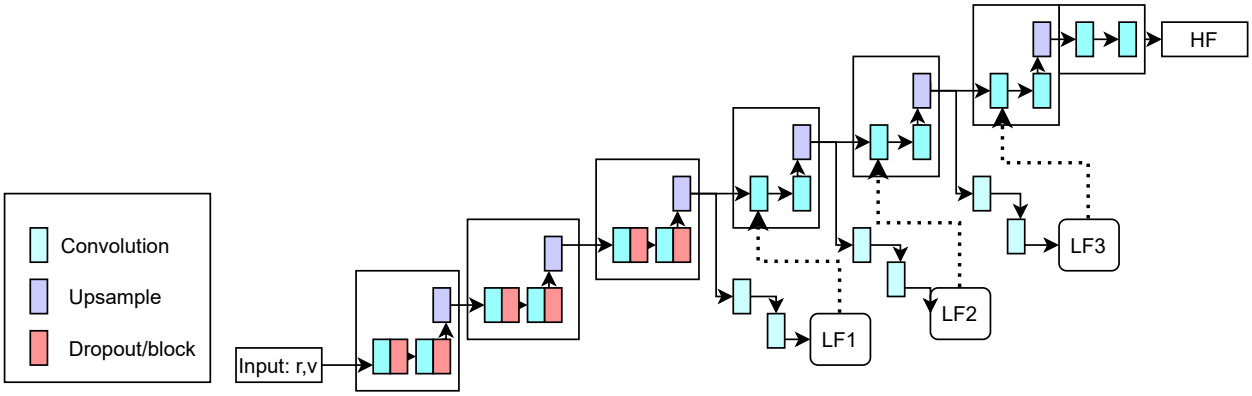


Figure 5: Multifidelity convolutional decoder architecture for low- to high-dimensional regression.

3.4 DropBlock layers for regularization and prediction uncertainty

Dropout layers [13] are a widely used form of regularization for artificial neural networks, designed to avoid overfitting. These layers operate by dropping neurons at random during training, so the learned representation relies only weakly on the correlations between neurons. This continuous change in network connectivity can be interpreted as the simultaneous training of an ensemble of architectures at a fraction of the computational cost of processing them individually. However, dropout layers are ineffective for convolutional neural networks due to the spatial correlation present in images, where relevant details consist of multiple correlated pixels. For this reason, DropBlock [16] layers are designed to drop a continuous group of pixels.

As these layers are still parameterized in terms of drop probability p, the relation between p and the actual ratio of features being dropped should be first clarified. A Bernoulli mask is generated in [16], using a probability γ expressed as

$$\gamma = \frac{pF^d}{b^d(F-b+1)^d}\,, \tag{7}$$

where p is the drop probability, F and b are the feature and block size, respectively, and d is the feature space dimensionality. Note that the drop probability p may not represent the actual percentage of elements dropped due to overlapping between blocks. As discussed in the appendix, the actual drop ratio is closest to p when $F = b$ (assuming no partial blocks are considered at the edges of the feature map); otherwise, we typically see a lower drop ratio due to overlapping blocks. In [16], a distinction is made between the DropBlock mask being independent or shared across feature channels; we considered both approaches and chose the one producing the best accuracy for each test case; we also compared the implications of both choices in Section 4.4.

When used during the *evaluation* of an optimally trained network, DropBlock layers can also be used to *inject stochasticity* in the network predictions, and, combined with Monte Carlo sampling, provide a tool to quantify output uncertainty. We refer to this technique as *MC DropBlock*, which is similar, in principle, to the MC dropout approach discussed in [15]. In this study, we consider network output *ensembles* of size $N_{UQ} = 1000$. As noted in Section 1, it is important to emphasize that this notion of uncertainty reflects the variability introduced in the network by the hyper-parameters (in this case changes in the network architecture induced by randomly dropping groups of features) and not the impact of any uncertainty either in the network weights (as in Bayesian neural networks, see [4, 24]) or its inputs.

We also wanted our network to promote accuracy in *each* MC-DropBlock realization rather than only on their mean. To do so, we activated DropBlock layers both in training and when *evaluating* the network (e.g. when calculating the validation loss). This is in contrast with the practice of keeping these layers off (i.e., drop probability $p = 0$) when generating network outputs, commonly adopted when using DropBlocks for mere regularization purposes. Keeping $p = 0$ during network evaluation leads to accurate *mean* predictions, but nothing prevents a single dropout sample from being inaccurate or having large oscillations, resulting in a significant increase in the prediction uncertainty. Except

for DropBlock layers, no other form of regularization was used for all the networks discussed in this work. Additional details on the implementation and hyperparameter selection for DropBlock layers are reported in the appendix.

3.5 High- to low-fidelity representation coupling

For each of these three networks, we consider an *implicit* and an *explicit* coupling between the LF and the HF representations. In the first implicit case, the LF predictors are not directly propagated towards the network output, as shown in Fig. 3, 4 and 5, where the black dotted arrows are omitted. However, forcing the upstream stages to learn accurate coarse pressure representations clearly affects the accuracy of the high-fidelity prediction. Propagation of information through the dotted arrows is instead allowed for the explicit feedback mechanism, meaning that the LF predictions are propagated through the following stages of the decoder. When the HF and LF truths belong to the same space and are correlated, an explicit connection helps in capturing the relationship between the LF and HF, such as in the two one-dimensional regression problems discussed in Section 2.1. However, it is unclear that an explicit connection would be beneficial for dense and low- to high-dimensional regression, since the LFs and HF live on different spaces. Since the LFs are of lower dimensionality than the HF, no bijective mapping exists between their respective spaces. In such a case, as it will be discussed in Section 4.2, our results seem to indicate the tendency of the network between two successive LFs to learn a *discrepancy* between their corresponding feature maps, ultimately leading to improved HF predictions.

4 Results

4.1 One-dimensional regression

We first trained the network only with LF data, to ensure there was no detriment to using a convolutional network as opposed to a fully-connected network (as in [3]) for one dimensional regression. After obtaining accurate predictions for the LF functions, we focused on multifidelity datasets. The explicit network outperformed the implicit network, which seems reasonable given the significant correlation between the LF and HF models.

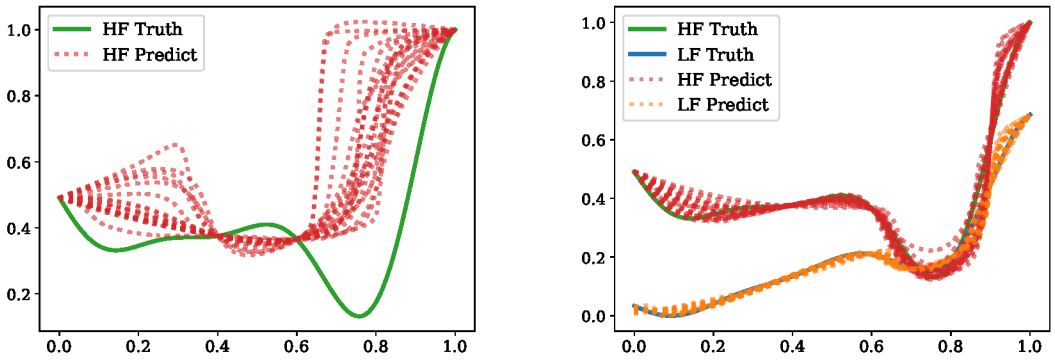
The multifidelity network is able to correctly leverage the LF data to influence the HF prediction to more closely resemble the true HF function, as shown in Fig. 6(b), compared to the predictions from the network trained with HF data only in Fig. 6(a). Similarly, in Fig. 7, the multifidelity network is able to capture the discontinuity by extracting this information from the LF data, since this feature could not be learned from the limited HF data. Equally accurate predictions in Fig. 6(a) and Fig. 7(a) result from different initial choices for the weights and biases (see appendix).

Including the spatial coordinate x as an additional input downstream of the LF predictor was a necessary adjustment needed to separate the LF into a linear and a non linear contribution, facilitating their combination into an optimal HF predictor; in this regard, note that in Eq. (4), $y_H(x) = \alpha y_L(x) + F_l(x)$, where α is a constant and $F_l(x) = -20x + 20$ is linear in x.

Although similar, the two problem sets for one-dimensional regression differ in one important aspect, i.e., the x coordinate for the HF samples is shared across fidelities for Eq. (1) and (2), whereas these locations are different for Eqs. (3) and (4). In this latter case, and in the absence of sufficient regularization, spikes may appear in the multifidelity network predictions at the locations of the HF data, produced by the LF predictor without altering the loss at the LF training locations (see, e.g., Fig. 16). Inclusion of multiple DropBlock layers provide sufficient regularization to prevent this behavior. This does not happen when using the same x values for the LF and HF datasets, since a spike reducing the HF loss would necessarily increase the LF loss.

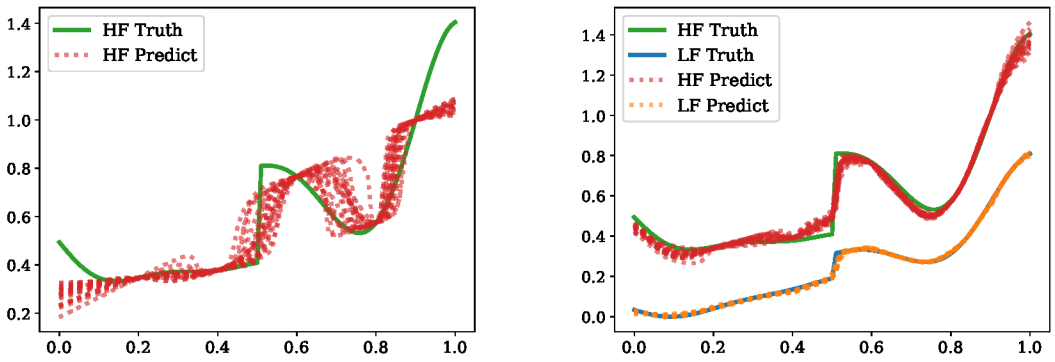
Finally, although [4] reported robust results, their network required the regularization penalty and the network size to be carefully selected to capture the true underlying HF-LF correlation. Since no validation set was included, choosing the regularization penalty and network size would require some degree of manual tuning. This operation might not be possible in a realistic application, since HF data may not be readily available. Therefore, the sensitivity of our network to the regularization penalty or other forms of regularization (e.g. DropBlock), for the example in Fig. 1(b), does not appear to be a limitation of this specific architecture.

In principle, the nonlinear convolutional sub-network in Fig. 3 could also capture linear correlations, under sufficient regularization. Under limited data, a linear kernel enforces this regularization without tuning regularization penalties to ensure that the simplest relationship is captured between the LF and HF. Additionally, for datasets with both nonlinear and linear correlations, excessive regularization on the kernels between the LF and HF predictors would result in only capturing the linear correlation. For the given set of hyperparameters, removing the linear kernel produces significantly less accurate HF predictions outside of the training points; however, decreasing the number of kernels in the nonlinear correlation results in accurate HF predictions even in the absence of the linear kernel.



(a) (b)

Figure 6: Mean predictions from (a) HF only network, (b) MF network, for different initial weights. True function values are from Eqs. (1)-(2).



(a) (b)

Figure 7: Mean predictions from (a) HF only network (b) MF network trained with LF and HF data, for different initial weights. True function values are from Eqs. (3)-(4)

4.2 Dense regression

We trained the multifidelity networks with implicit and explicit feedback using 32 HF and 116 LF pressure samples for each resolution (this network configuration is denoted as *MF 32/116*), and compared its outputs with those from the HF 116/0 and HF 32/0 networks, respectively.

The multifidelity network with implicit feedback significantly improves the accuracy with respect to the HF 32/0 network, and its normalized accuracy is also superior to the HF 116/0 network. In the case with multiple drop layers, the explicit multifidelity network accuracy suffers in the case of additive skip connections. High-fidelity and multifidelity validation loss profiles are shown in Fig. 8. Fig. 8(a) compares the HF contribution to the validation loss for four networks, i.e., HF 116/0, HF 32/0 and MF 32/116 with both implicit and explicit multifidelity coupling. The plot emphasizes the acceleration in convergence produced by training networks with multifidelity data for the chosen hyperparameters in the implicit case; however, this may not be consistent across all sets of hyperparameters. Figure 8(b), on the other hand, demonstrates the ability of the network to learn multiple LF representations during training.

The approximation accuracy of the HF 32/0 and MF 32/116 networks is compared for two pressure examples from the test set in Fig. 9 and for slices of the pressure field parallel to the cylinder generator in Fig. 10. The multifidelity network shows more consistently accurate predictions across the entire image instead of a localized region. The larger pressure errors noticeable near the fluid boundary appear typical of convolutional neural networks, which often report

lower accuracy near the boundary (see, e.g., [25]). This effect is magnified in our case as the boundary, which is associated with the (random) diameter, changes with every sample. The original U-Net architecture overcomes this through reflection padding on the input layer and no padding on any subsequent layers [22] (whereas we zero pad each convolutional layer), although other approaches have been proposed in the convolutional network literature to overcome this limitation (see, e.g., [25]).

4.2.1 Effect of bias in low-fidelity predictor

To explore to some extent the limitations of this network with regards to the accuracy of the low-fidelity data, we perform an additional test where the LF3 is biased. We choose the LF3 since we expect this data to have the most significant effect on the HF prediction due to its proximity in the network. Specifically, we add a constant bias $r \cdot [\max_{j,k} \mathrm{LF}_3(j,k) - \min_{j,k} \mathrm{LF}_3(j,k)])$ where the bias ratio is $r = 0.01, 0.05$, to all of the LF3 data. We first use the hyperparameters chosen in Table 6. As shown in Table 2, most of the accuracy values are similar to those found in the absence of biased low-fidelity data (i.e. Table 1), except a single outlier for the MF additive explicit feedback. However, after the selection of more appropriate hyperparameters, the accuracy for this test case is again similar to the case without bias (see updated accuracy in parenthesis). This seems to suggest the ability of the network to counteract excessive bias or inaccuracy in the LF predictors, but further analysis is required.

Skip Conn.	Network Type	MF Feedback	HF/LF	R^2	Normalized R^2
Concat	MF	Explicit	32/116	0.9326	3.250e-06
Add	MF	Explicit	32/116	0.9372	3.266e-06
Concat	MF	Implicit	32/116	**0.9672**	3.370e-06
Add	MF	Implicit	32/116	0.9541	3.325e-06
Concat	HF	-	32/0	0.9284	**7.083e-06**
Add	HF	-	32/0	0.9047	6.902e-06
Concat	HF	-	116/0	0.9408	1.980e-06
Add	HF	-	116/0	0.9257	1.948e-06

Table 1: Comparison of HF and MF network performance for dense regression. The normalized accuracy is R^2/C where C is the cost (see Section C.2). The terms *Concat* and *Add* refer to how the information from a skip connection is assembled into the decoder.

4.3 Low- to high-dimensional dense regression

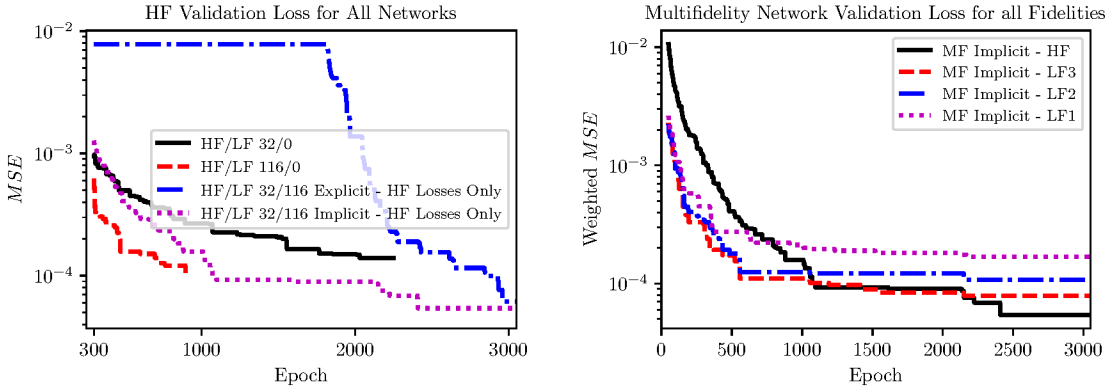
The results from the low- to high-dimensional decoder architecture reported in Table 3 show competitive test set accuracy for the MF 32/116 network with respect to the HF 116/0 network. In addition, both MF networks perform better than the HF 32/0 network, as shown in Fig. 11.

4.4 Uncertainty quantification of network predictions

In this section, we analyze the uncertainty estimates from MC DropBlock, or, in other words, we quantify the variability from an ensemble of predictions obtained by feeding the same input to the network $N_{\mathrm{UQ}} = 1,000$ times.

Bias ratio	Skip Conn.	Network Type	MF Feedback	HF/LF	R^2
0.01	Concat	MF	Explicit	32/116	0.944853
0.05	Concat	MF	Explicit	32/116	0.933174
0.01	Add	MF	Explicit	32/116	0.198742 (0.935613*)
0.05	Add	MF	Explicit	32/116	0.92367
0.01	Concat	MF	Implicit	32/116	0.95079
0.05	Concat	MF	Implicit	32/116	**0.960783**
0.01	Add	MF	Implicit	32/116	0.942729
0.05	Add	MF	Implicit	32/116	0.949461

Table 2: Accuracy of multifidelity networks in Table 1, when trained with biased LF3. (*) Validation accuracy obtained by re-optimizing the network hyperparameters.



(a) (b)

Figure 8: Validation loss profiles for dense pressure prediction in Poiseuille flow (a). The plot compares the losses resulting from HF only and MF training, using the best model for each category, i.e. with the highest test accuracy of those shown in Table 1. The profiles for the weighted mean squared losses integrated over the fluid domain are shown in (b) for the best performing MF approach. All networks are trained for the same number of epochs, but only the decreasing losses are shown. Losses are plotted on a log scale.

Network Type	**MF Feedback**	**HF/LF**	R^2	**Normalized** R^2
MF	Explicit	32/116	**0.9330**	3.251e-06
MF	Implicit	32/116	0.9054	3.155e-06
HF	-	32/0	0.8369	**6.385e-06**
HF	-	116/0	0.8907	1.875e-06

Table 3: Low- to high-dimensional regression comparison of HF and MF network performance. Accuracy is computed similar to Table 1.

4.4.1 One-dimensional regression

We experimented with different drop probability schedulers which have been shown to produce improved testing accuracy, including an exponential decay [26] and a delayed start to a linear scheduler. No scheduler was found to generate accurate mean predictions and, at the same time, uncertainty intervals which bounded both the true LF and HF functions. Exponential schedulers were found to produce less accurate dropout realizations and wider uncertainty intervals. Additionally, no significant difference was observed by changing the speed at which the linear drop probability scheduler ramps up to the desired p (see Fig. 12). Therefore, for the results reported in Figs. 13 and 15, no drop probability scheduler was applied.

Large uncertainty intervals resulted by adding a DropBlock layer after each convolutional layer, as shown in Fig. 13 and Fig. 15. However, this generated inaccurate individual realizations, due to excessive regularization. We then investigated the possibility to selectively remove DropBlock layers in order to minimize the mean square error over all DropBlock realizations for all training examples, with results shown in Fig. 14 and Fig. 16. As expected, the uncertainty intervals are extremely narrow at the training points. In between the training points, however, we see varying levels of uncertainty. Although the uncertainty does not always capture the underlying true response, it provides an indication of where data should be selected to improve the network accuracy. For example, in Figs. 14(a) and 14(b), we see the largest uncertainty near $x = 0.05$, which is also where we see the largest error in the predictions. Similarly, in Fig. 16(b), we see large uncertainty near $x = 0.45$, $x = 0.55$ surrounding the discontinuity.

Uncertainty bounds predicted using ReLU or tanh activation functions appear similar when multiple Dropblocks are introduced in the network, and maximum uncertainty is achieved at the same values of x, as shown in Fig. 17 and 18. Slightly smaller standard deviations are produced by ReLU due to its greater flexibility (due to scale invariance and lack of saturation) and since single DropBlock realizations are included in the loss function. Moreover, we observe almost identical patterns for the standard deviation in the LF and HF functions, as expected. Since all DropBlock layers are

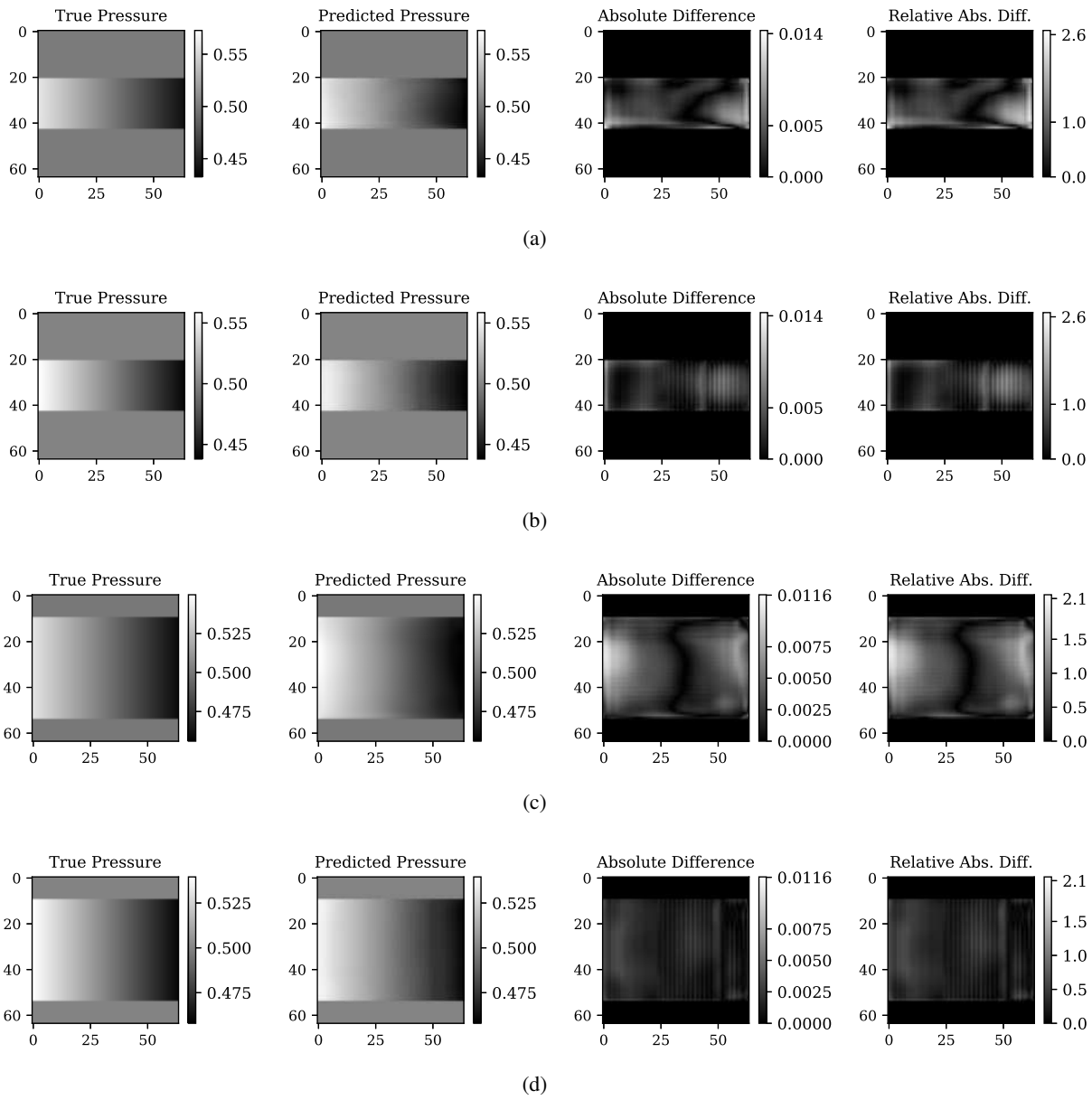


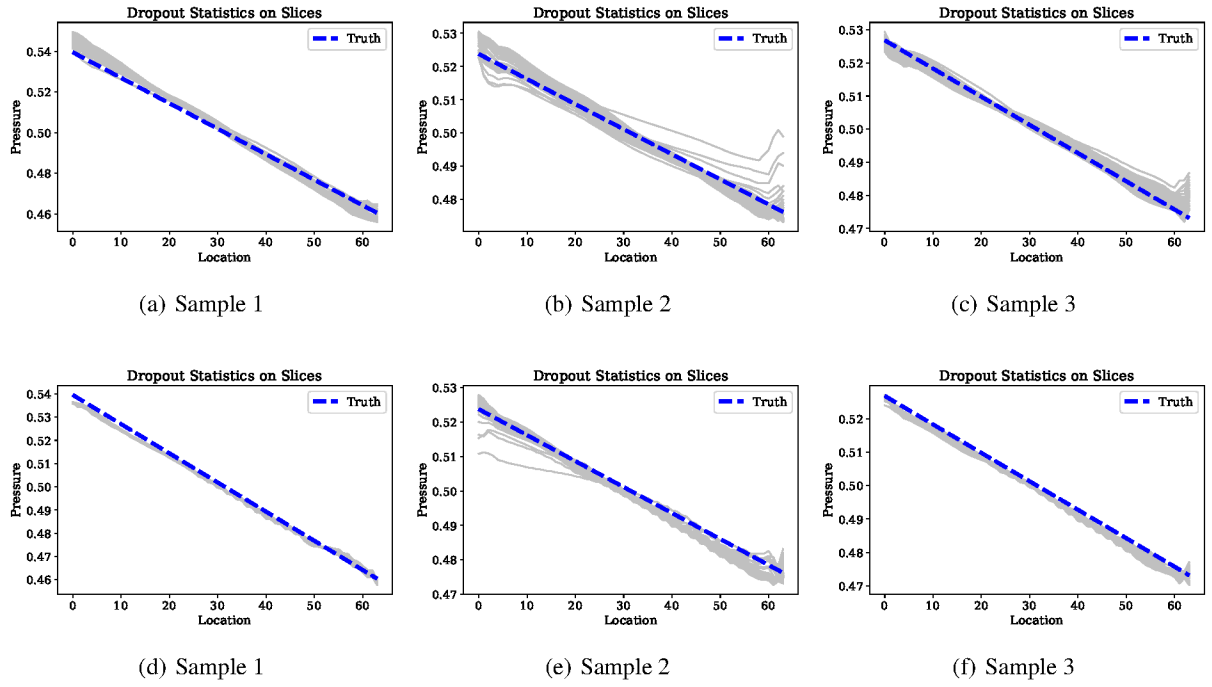
Figure 9: Dense regression network. Mean predictions from HF 32/0 on two pressure configurations from the test set (a,c). Mean predictions from MF 32/116 on the same test examples (b,d). Both network setups lead to accurate predictions with limited absolute and relative errors.

located before the LF prediction, HF prediction uncertainty is propagated identical from the LF representation. ReLU tends to spike to a larger value within a small interval, which may relate to its unbounded property.

Addition of LF training data leads to a reduction in the estimated uncertainty, but preliminary tests suggest a re-calibration of the hyperparameters to be essential for this to occur.

4.4.2 Dense regression

Uncertainty estimates for the pressure along the centerline of the fluid region from N_{UQ} realizations are shown in Fig. 19. As is most evident in Fig. 19(b), we see that the HF 32/0 produces less accurate predictions and wider uncertainty intervals.



(a) Sample 1 (b) Sample 2 (c) Sample 3

(d) Sample 1 (e) Sample 2 (f) Sample 3

Figure 10: Dense regression network. Pressure predictions of test data on multiple slices parallel to the cylinder generator in the fluid region. Note that more slices are available for samples associated with a cylinder of larger radius r. (a)-(c) Mean predictions from HF 32/0 network with highest test accuracy. (d)-(f) Mean predictions from MF 32/116 network with highest test accuracy.

In Fig. 20 we plot the mean square error (MSE) and standard deviation of N_{UQ} MC-DropBlock realizations for every example in the test set, and plot them versus the corresponding location along the centerline of the cylindrical fluid domain. Both the variance and the MSE appear parabolic, consistent with the pressure results being approximately 0.5 for all samples at the center of the fluid domain, due to the way samples are normalized. Also, error and uncertainty increase away from the center due to the slope variability in the true pressure profiles. MF networks are observed to produce lower errors and limited uncertainty with respect to HF networks. The HF 32/0 is characterized by both the highest variance and highest error, which is consistent with the limited amount of data available during training.

We also compare the results from DropBlock masks being shared or independent across feature channels. The study in [16] concludes that independent DropBlock masks work better, but it only analyzes the resulting network accuracy and not prediction uncertainty. Our results in Fig. 21 show comparable accuracy for shared and independent masks.

4.4.3 Low- to high-dimensional regression

In Fig. 22 we plot the mean square error (MSE) and standard deviation of the MC-DropBlock realizations across test samples versus their location along the axis of the cylindrical fluid domain. Similarly to Section 4.4.2, the MF network produces results as accurate as the HF 116/0 network, while also producing lower variance than the HF networks. Some of this difference could be attributed to different drop probabilities, since higher drop probability tended to produce higher variance in our experiments. Both the variance and MSE appear parabolic, consistent with the conclusions presented in Section 4.4.2.

5 Conclusions and future work

In this work, we focus on convolutional neural networks, specifically architectures resulting from an assembly of encoders, decoders and skip connections. Such architectures have the flexibility to predict the results from HF physics-based solvers having either high-dimensional inputs, high-dimensional outputs or both, using only a fraction of the weights that would be required by fully connected networks. If trained from a few expensive HF and many inexpensive LF examples, they also exhibit a comparable performance to fully-connected multifidelity networks for

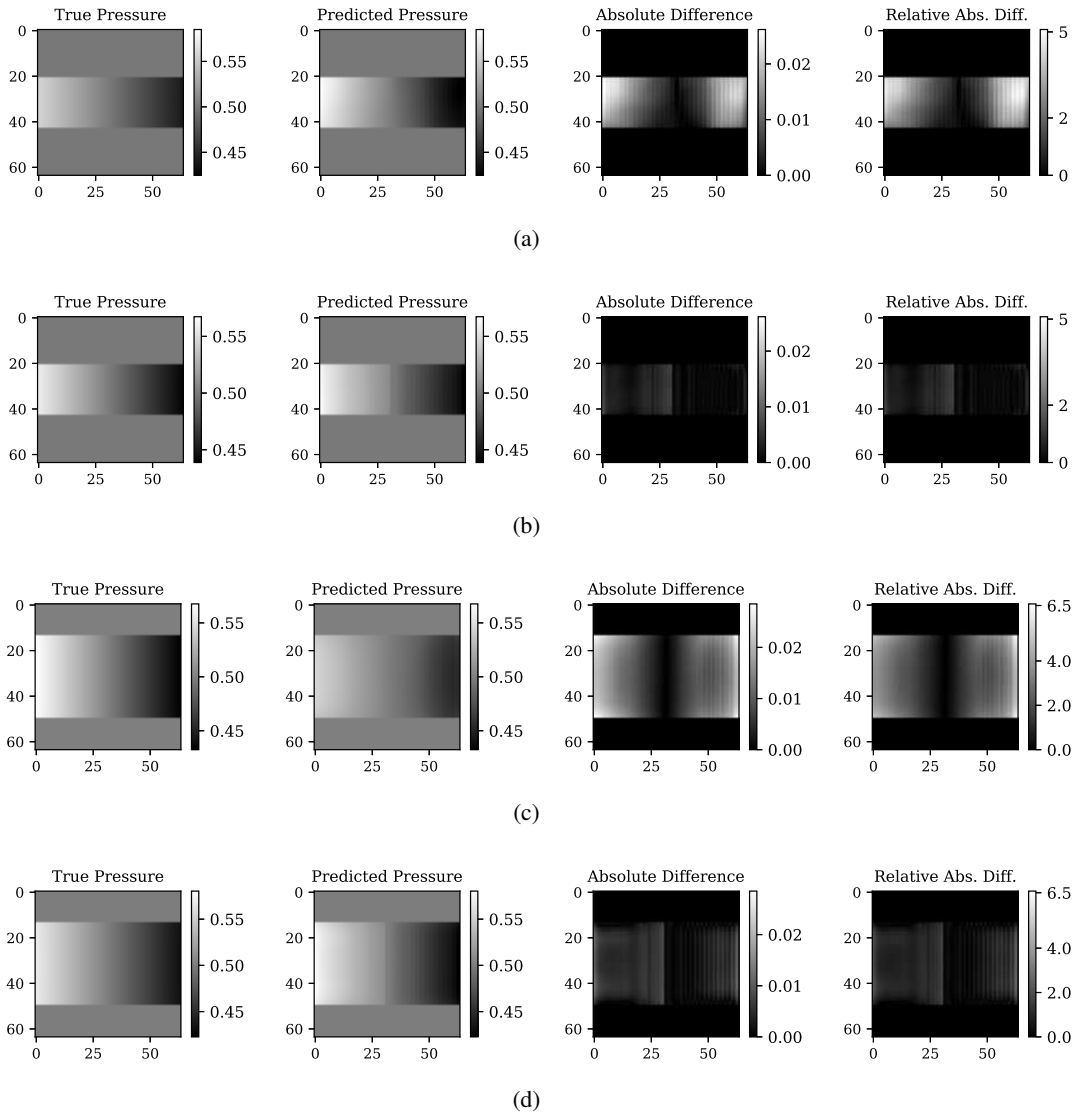
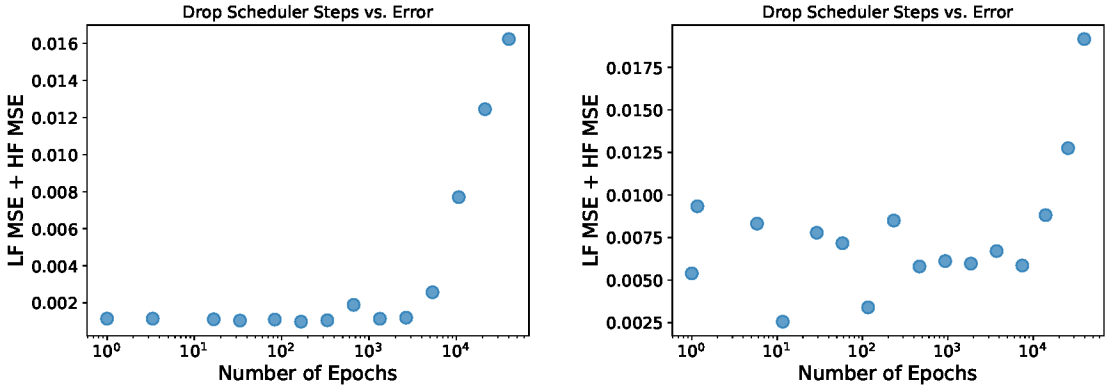


Figure 11: Predictions from the decoder network for the low- to high-dimensional regression test case. (a,c) Mean prediction from HF 32/0 network. (b,d) Mean prediction from MF 32/116 network.

one-dimensional function approximation and produce a consistently accurate performance for high-dimensional inputs/outputs.

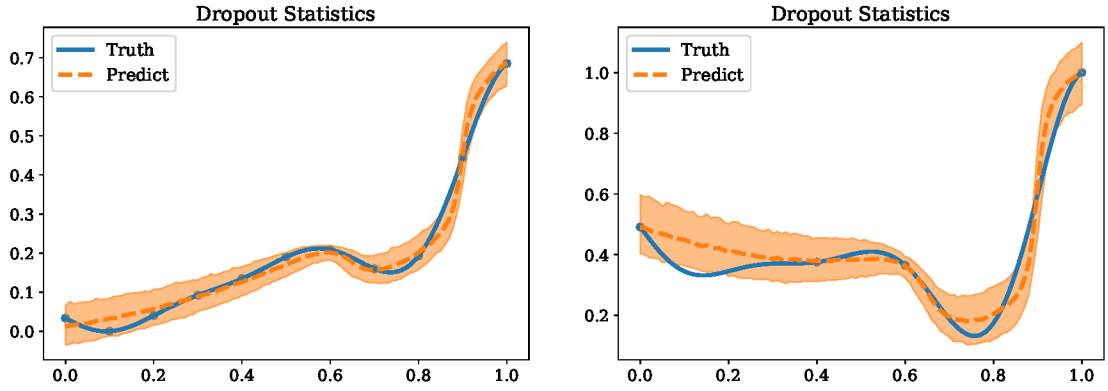
In this context, using two test cases in one-dimensional functional approximation and the solution of the pressure Poisson equation, respectively, we show that multifidelity networks produce, at a reduced cost, a validation accuracy comparable to that of networks trained from a much larger number of high-fidelity realizations. Use of datasets containing examples from multiple fidelities also accelerates training, leading to significant loss reductions early on during the training process.

We also focus on quantifying the variability in the network predictions using DropBlocks. Using DropBlock layers not only during training, but also during testing and validation (and also investigating the impact of their locations and count) improves the accuracy of each MC-DropBlock realization, leading to more robust uncertainty estimates and reduced variability. We also investigated how the location and the number of DropBlock layers affect prediction uncertainty. Adding multiple DropBlocks after each convolutional layer, while still providing a shared parameterization across all fidelities, appear to maximize the uncertainty due to variability in the network architecture. However, this was found to produce excessive regularization and reduced accuracy for the decoder-encoder network used for



(a) (b)

Figure 12: One-dimensional regression errors produced by varying the drop probability scheduler ramp. Mean squared error (MSE) is computed over all dropout predictions for the LF training points and HF training points separately, and summed. The x axis refers to the number of epochs required to reach $p = 0.1$ using a linear drop probability scheduler. (a) Eq. (1)-(2) with dropout layer after every convolution layer, *except* the first two convolution layers, preceding the LF prediction and (b) Eq. (3)-(4) with dropout layer after every convolution layer, *except* the first four convolution layers, preceding the LF prediction. A tanh activation is used in all these tests.



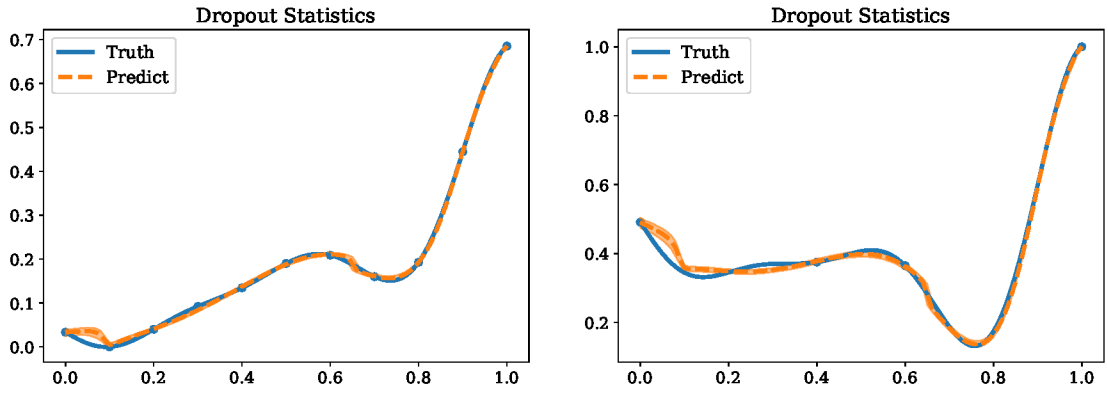
(a) (b)

Figure 13: Dropout layer after every convolution layer preceding the LF prediction. Mean prediction and estimated 5%-95% percentiles for (a) Eq. (1) and (b) Eq. (2), from a network with multiple DropBlock realizations. In these tests we used a tanh activation function.

one-dimensional regression. DropBlock masks that are independently applied or shared across feature channels were found to produce similar accuracy and uncertainty estimates, as well as the use of ReLU or tanh activation functions.

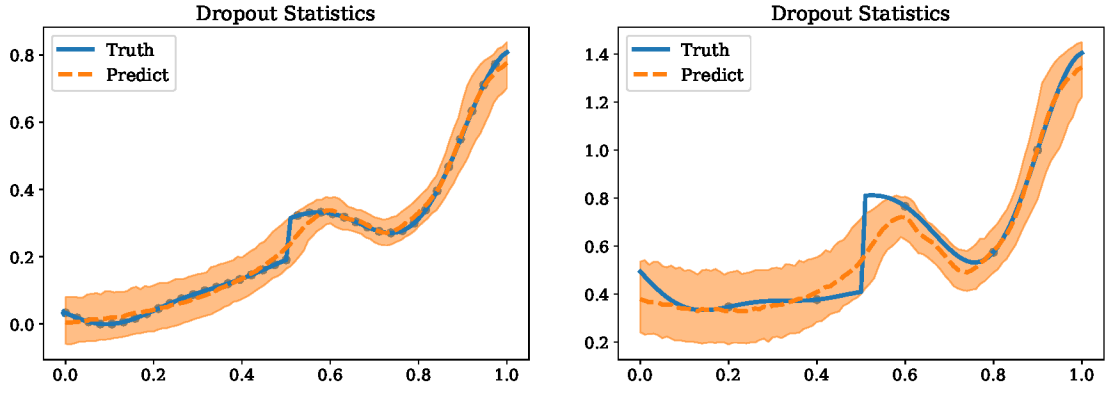
This proposed approach is not without limitations. First, all the results presented in this paper are based on optimal hyperparameters selected through a systematic grid search. Improved results could in principle be obtained from continuous parameter optimization. Also, inclusion of a large number of LF models, possibly non ordered hierarchically, in the proposed network would require some architectural changes. This could be realized by increasing the depth (number of downsampling blocks) of the network, assuming the input resolution is greater than 2^{depth}. In addition, the order of the LF predictors could reflect other aspects besides resolution, such as which LF has the highest correlation with the HF or the highest accuracy.

Future work will be devoted to investigating the performance of additional network layouts, testing new encoder-decoder configurations and on comparing the variance reduction of MC-DropBlock with multifidelity Monte Carlo estimators like the ones covered in [27].



(a) (b)

Figure 14: Dropout layer after every convolution layer, *except* the first two convolution layers, preceding the LF prediction. Mean prediction and estimated 5%-95% percentiles for (a) Eq. (1) and (b) Eq. (2), from a network with multiple DropBlock realizations. In these tests we used a tanh activation function.



(a) (b)

Figure 15: Dropout layer after every convolution layer preceding the LF prediction. Mean prediction and estimated 5%-95% percentiles for (a) Eq. (3) and (b) Eq. (4), from a network with multiple DropBlock realizations. In these tests we used a tanh activation function.

A Network implementation details

A.1 One-dimensional regression

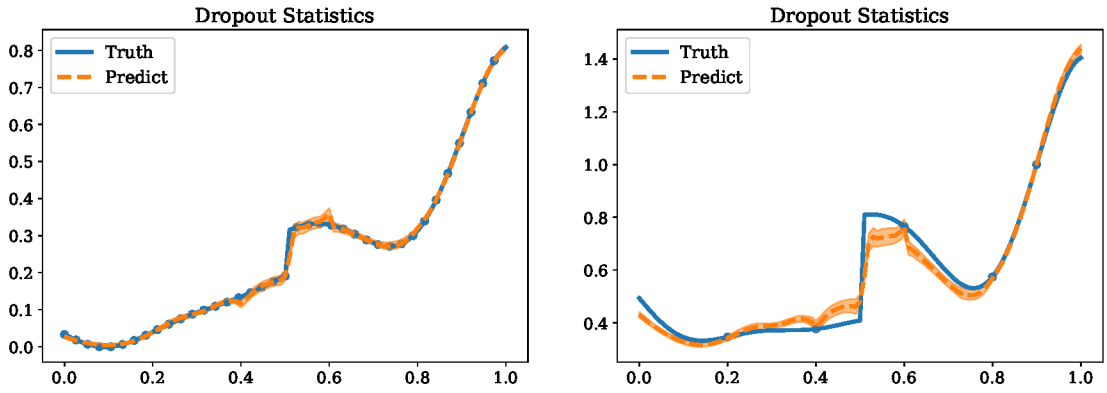
The architectural details for this network are reported in Table 4.

A.2 Dense regression

Additional details on the architecture of the dense regression network are reported in Table 5.

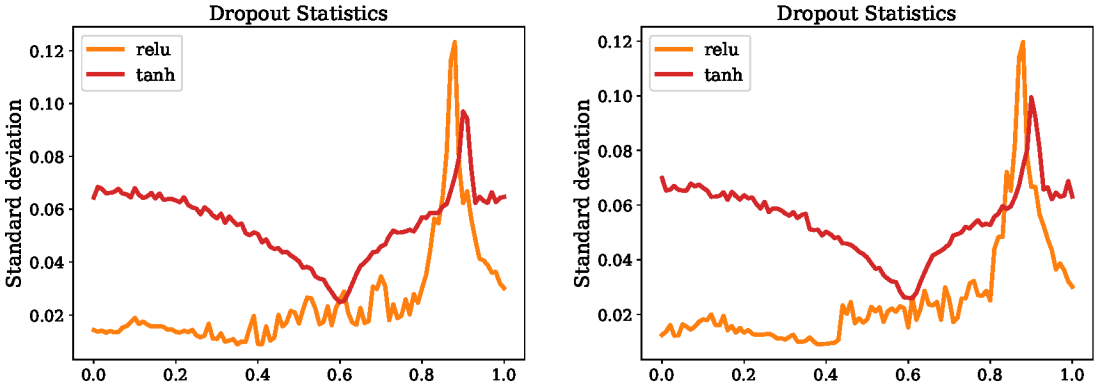
A.3 Low-to-high dimensional regression

The number of channels input to the final convolutional layer is defined in the last column of Table 9, where preceding layers increase by a factor of 2 (e.g. for a value of k, the number of kernels per convolution layer would be $k \times 2^5, k \times 2^5, k \times 2^4, k \times 2^4, k \times 2^3, k \times 2^3, k \times 2^2, k \times 2^2, k \times 2^1, k \times 2^1, k \times 2^0, k \times 2^0, k \times 2^0, 1$). The number of kernels for the two convolutions immediately preceding the low-fidelity outputs are the same as the two kernels immediately



(a) (b)

Figure 16: Dropout layer after every convolution layer, *except* the first four convolution layers, preceding the LF prediction. Mean prediction and estimated 5%-95% percentiles for (a) Eq. (3) and (b) Eq. (4), from a network with multiple DropBlock realizations. In these tests we used a tanh activation function.



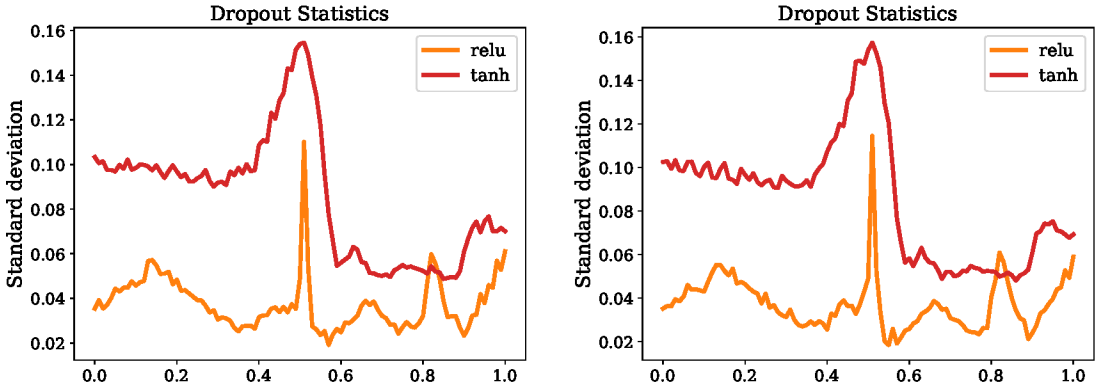
(a) (b)

Figure 17: Tanh vs. ReLU activation functions comparison of standard deviation from multiple DropBlock realizations. Dropout layer after *every* convolution layer preceding the LF prediction. (a) LF Eq. (1) and (b) HF Eq. (2).

Component	Layers	Kernels per convolution layer	Kernel
Encoder	2 CBTD-CBTD-U layers	16 16 8 8	2 2 1 1*
Decoder	2 CBTD-CBTD-M layers	8 8 16 16 1	1 1 2 2*
Nonlinear correlation	1 CBT-C layer	8 1	2 1*
Linear correlation	1 C layer	1	1*

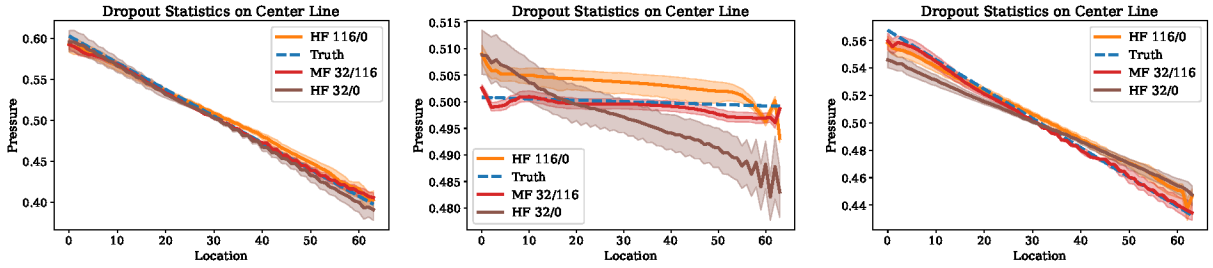
Table 4: Architectural details for one dimensional regression network (see Figure 3). Layers consist optionally of various components including convolution (C) layer, batch normalization (B) layer, ReLU activation (R), tanh activation (T), max pooling (M) layer, upsampling (U) layer, and DropBlock/dropout (D) layer. (*) a stride of 1 and appropriate padding are used on all convolution layers to produce layer inputs and output of the same size.

preceding the HF prediction. Hyperparameters are shown in Table 9. Otherwise, we use the same hyperparameters specified in Section A.2.



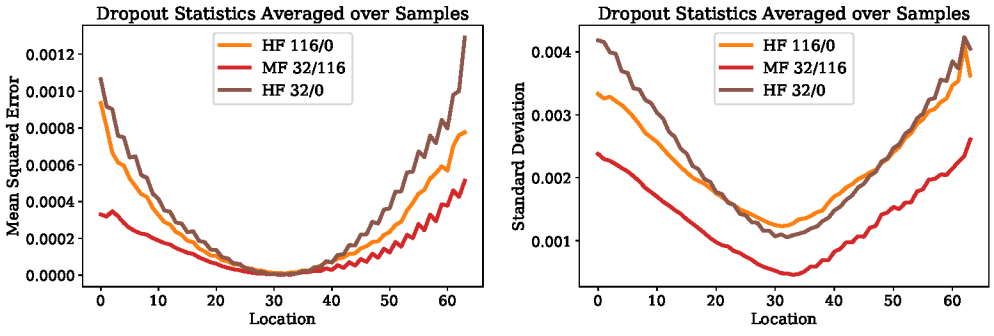
(a) (b)

Figure 18: Tanh vs. ReLU activation functions comparison of standard deviation from multiple DropBlock realizations. Dropout layer after *every* convolution layer preceding the LF prediction. (a) LF Eq. (3) and (b) HF Eq. (4).



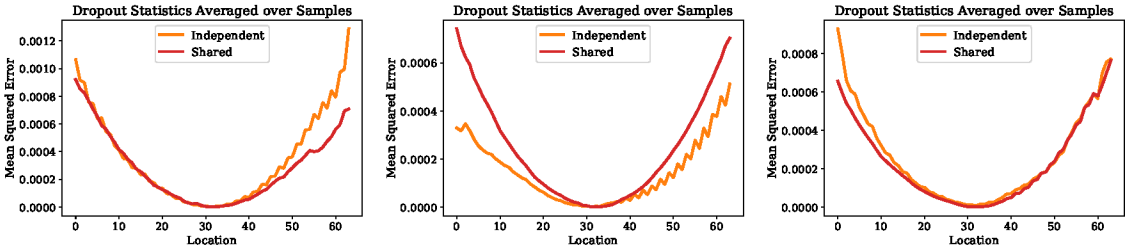
(a) Sample 1 (b) Sample 2 (c) Sample 3

Figure 19: Dense regression network. Mean prediction and 5%-95% percentiles from an ensemble of 1000 DropBlock pressure realizations for three test data examples sliced along the cylinder centerline.

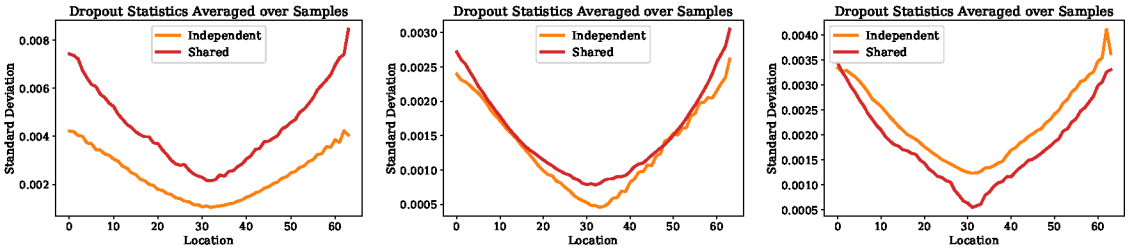


(a) (b)

Figure 20: Dense regression network. Mean square error and standard deviation resulting from 1000 network evaluations with 10 DropBlock layers in each network. Quantities are calculated along the axis of the cylindrical fluid domain. The optimal network is selected using the minimum validation loss calculated by activating all DropBlock layers.

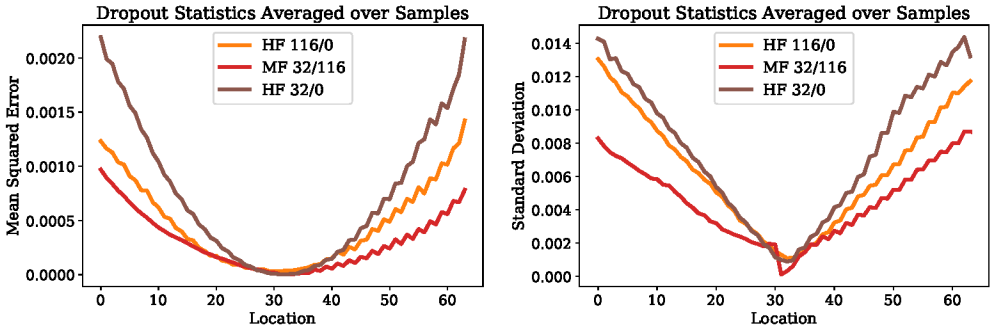


(a) HF 32/0 (b) MF 32/116 (c) HF 116/0



(d) HF 32/0 (e) MF 32/116 (f) HF 116/0

Figure 21: Dense regression network. Mean squared errors (top row) and standard deviations (bottom row) from 1000 DropBlock realizations along the axis of the cylindrical fluid domain for networks containing 10 DropBlock layers, The optimal network is selected using the minimum validation loss calculated by activating all DropBlock layers.



(a) (b)

Figure 22: Low- to high-dimensional case: Mean square error and standard deviation resulting from 1000 network evaluations with 6 DropBlock layers in each network. Quantities are calculated along the axis of the cylindrical fluid domain. The optimal network is selected using the minimum validation loss calculated by activating all DropBlock layers.

Component	Layers	Kernels per convolution layer	Kernel
Encoder	4 CBRD-CBRD-M layers	16 16 32 32 64 64 128 128	3x3-s1-p1
Decoder	1 CBRD-CBRD-U, 3 CBR-CBR-U, 2 C layers	64 64 32 32 16 16 32 32 16 1	3x3-s1-p1
Low-fidelity outputs	CBR-C	16 1	3x3-s1-p1

Table 5: Number of convolutional layers and channels for dense regression network in Figure 4. The notation *nxn-sm-pk* indicates a square kernel of size n with stride of m pixels and padding of k pixels.

Fidelity	Skip	LR	LR scheduler steps	Drop scheduler steps	Drop probability
Explicit	Concat	1×10^{-2}	200	None	0.1
Explicit	Add	5×10^{-2}	200	None	0.3
Implicit	Concat	1×10^{-2}	500	300	0.1
Implicit	Add	1×10^{-2}	1000	300	0.3
HF 32/0	Concat	2×10^{-2}	1000	None	0.3
HF 32/0	Add	1×10^{-2}	1000	None	0.1
HF 116/0	Concat	5×10^{-2}	500	None	0.1
HF 116/0	Add	2×10^{-2}	200	300	0.1

Table 6: Dense regression network. Hyperparameters producing the best validation accuracy on the mean prediction for each HF and MF model, for the case where DropBlocks are independent across channels.

Fidelity	Skip	LR	LR scheduler steps	Drop scheduler steps	Drop probability
Explicit	Concat	1×10^{-2}	200	None	0.1
Explicit	Add	1×10^{-2}	200	300	0.3
Implicit	Concat	1×10^{-2}	500	None	0.1
Implicit	Add	1×10^{-2}	200	300	0.5
HF 32/0	Concat	2×10^{-2}	500	300	0.3
HF 32/0	Add	1×10^{-2}	200	None	0.5
HF 116/0	Concat	1×10^{-2}	1000	300	0.3
HF 116/0	Add	1×10^{-2}	200	300	0.5

Table 7: Dense regression network. Hyperparameters producing the best validation accuracy on the mean prediction for each HF and MF model, for the case where DropBlocks are shared across channels.

Component	Layers	Kernels per convolution layer	Kernel
Decoder	3 CBRD-CBRD-U, 3 CBR-CBR-U, 1 CBR-C	128 128 64 64 32 32 16 16 8 8 4 4 4 1	3x3-s1-p1
Low-fidelity outputs	CBR-C	4 1	3x3-s1-p1

Table 8: Number of convolutional layers and channels for low- to high-dimensional regression network in Figure 5. The notation *nxn-sm-pk* indicates a square kernel of size n with stride of m pixels and padding of k pixels.

B Implementation details for DropBlock layers

We observed how the increased regularization produced by additional DropBlock layers could act as a valid substitute for ℓ_1 or ℓ_2 regularization applied directly to the loss function. This was found particularly useful for the dataset in Eqs. (3) and (4), characterized by a lack of overlap between the LF and HF training locations.

Some additional constraints have to be considered in the selection of suitable locations for DropBlock layers, in order to maintain a *shared parameterization* for all HF and LF outputs. We choose not to include DropBlock layers at locations which would induce stochasticity only to a subset of the HF or LF predictors, i.e. we omit DropBlock after any convolution layers following the LF network outputs. Consistent with these constraints, we apply the same binary mask both prior to the skip connection and prior to the pooling layer at each stage of the encoder of the network in Fig. 4.

B.1 One dimensional regression

We consider DropBlocks after the first eight convolution layers (see Fig. 3), since these layers precede the LF prediction. However, since these result in very inaccurate predictions, we exclude as few of the dropout layers as results in accurate dropout realizations; for the dataset in Eq. (2), we use DropBlock layers 3-8 and for the dataset in Eq. (4), we use DropBlock layers 5-8 (Fig. 3). For the DropBlock layers, we use a drop probability of 0.1 and a block size of 1, due to the low dimensionality of the layers. A block size of 1 would be equivalent to a dropout layer, since $\gamma = p$ from Eq. (7) and single pixels in the feature map are dropped independently. We use no scheduler for the drop

Fidelity	LR	LR scheduler steps	Drop scheduler steps	Drop probability	Filters
Explicit	2×10^{-2}	500	300	0.1	6
Implicit	1×10^{-2}	1000	None	0.3	4
HF 32/0	2×10^{-2}	500	None	0.5	5
HF 116/0	2×10^{-2}	1000	300	0.1	4

Table 9: Decoder network. Hyperparameters producing the best validation accuracy on the mean prediction for each HF and MF model.

probability. The DropBlock mask is independent across feature channels due to the small dimensionality of the layers. Each convolutional layer is followed by a *tanh* activation, with the exception of the final layer where we use a linear activation.

b	p	F	γ	Actual drop ratio	b	p	F	γ	Actual drop ratio
3	0.2	3	0.2	0.203	5	0.2	8	0.08	0.183
3	0.9	3	0.9	0.901	5	0.9	8	0.36	0.631
3	0.2	4	0.133	0.19	5	0.2	16	0.053	0.184
3	0.9	4	0.6	0.718	5	0.9	16	0.24	0.609
3	0.2	8	0.089	0.186	7	0.2	8	0.114	0.196
3	0.9	8	0.4	0.652	7	0.9	8	0.514	0.701
3	0.2	16	0.076	0.186	7	0.2	16	0.046	0.178
3	0.9	16	0.343	0.652	7	0.9	16	0.206	0.589

Table 10: Dropout probability vs. the average ratio of features dropped. This latter is computed across 1000 DropBlock realizations and averaged. This is evaluated for a one-dimensional layer, where there are 8 channels, and masks are independent across feature channels. The feature size F represents the number of dimensions in each channel. We exclude all partial blocks to be more consistent with [16].

B.2 Dense regression

We investigated DropBlock layers that are shared or independent across feature channels. Although shared masks were found to work well when only using a single DropBlock layer, in this work we focus on independent masks, with some uncertainty results related to shared masks. Additionally, DropBlock layers use a block size of 3, in combination with a linear scheduler for the drop probability, with parameters as specified in Table 6 and 7. DropBlock layers are included after the first ten convolution layers, preceding the LF1 output, as shown in Fig. 4.

B.3 Low-to-high dimensional regression

For low- to high-dimensional regression, we apply DropBlocks after the first six convolution layers of Fig. 5 preceding the LF1 output. We use block sizes of 1, 1, 1, 1, 3, 3, respectively, from the first to the last DropBlock layer and no linear scheduler for the drop probability.

C Additional implementation details

C.1 Hyperparameter search

The results in Section 4 are obtained using combinations of hyperparameters producing minimal validation losses. These hyperparameters include the learning rate, step size of the learning rate scheduler, number of filters, regularization penalty, weight initialization scheme, batch size, optimizer, DropBlock location and drop probability.

C.2 Accuracy metric

Prediction accuracy is evaluated on the test dataset and reported as the coefficient of determination R^2, within the true fluid region,

$$R^2 = 1 - \text{RSE} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{8}$$

where RSE is the relative squared error, y_i the true pixel value with mean $\bar{y}$, $\hat{y}_i$ the network mean prediction (mean over $N_{\text{UQ}} = 1,000$ DropBlock realizations), and N is calculated across the entire test set (i.e., the number of pixels in the fluid across all test samples). We also report a normalized accuracy with respect to the cost of generating the training data set, quantified, in this study, as the total number of pixels in the data. Thus, 116 HF images with resolution 64×64 have an equivalent cost of $475,136$ pixels, while 32 HF images have an equivalent cost of $131,072$ pixels (a cost ratio of 0.276 to the 116 HF case). The multifidelity dataset with 32 HF images and 116 images for each LF would instead result in a cost of $286,976$ pixels (a cost ratio of 0.604 to the 116 HF case).

C.3 One-dimensional network

C.3.1 Dataset preprocessing

We use the same training data as detailed in [3] and a test set consisting of 101 equally spaced points in the interval $[0, 1]$. Since a validation set does not exist, we also optimize for the best fit over the training set. In addition, the functions are rescaled to the range $[0, 1]$ based on the maximum and minimum value in the training set, which allows for a consistent use of the same optimizer across functions.

C.3.2 Regularization

In [8], we carefully selected the ℓ_2 regularization penalty to constrain the relationship between the LF and HF and therefore, in general, its value may depend on the dataset. However, in this paper, the use of extra DropBlock layers precluded the necessity of applying ℓ_2 regularization.

C.3.3 Weight optimization and batch size

Training is performed using the Adam optimizer [28] utilizing a step learning rate scheduler with decay 0.9, where the step size and initial learning rate is determined by optimizing for the best fit of the given training dataset. We use a learning rate of 9×10^{-4} and a step size of 450. A batch size of 1 was used, since gradient updates based on a small batch size were found to significantly improve prediction accuracy.

C.3.4 Weight initialization

The network weights are initialized from a a uniform distribution $U(-s, s)$, with $s = nk_0k_1$, n being the number of input channels, and $k = (k_0, k_1)$ being the kernel shape.

C.4 Two-dimensional networks

C.4.1 Dataset preprocessing

For the dense and low- to high-dimensional cases, the dataset consists of two-dimensional slices from a Hagen-Poiseuille flow in a cylindrical fluid domain Ω_f. The solution is axisymmetric and therefore equal with respect to any plane that includes the cylinder axis; therefore a two-dimensional slice is sufficient to fully describe the flow.

C.4.2 Single- and multifidelity dataset selection

The Hagen-Poiseuille flow dataset (denoted as *HF 116/0*) consists of 116 HF realizations corresponding to random values of the maximum velocity and cylinder radius parameters (see Fig. 2). Training, validation and testing datasets are obtained using 60/20/20 split ratios, resulting in 116, 49 and 35 HF images, respectively. For each sample, we generate a uniformly random floating point, which determines to which set that sample belongs based on their respective probabilities; therefore, exact ratios are not preserved. A second dataset (*HF 32/0*) was also generated by randomly subsampling 32 of the 116 HF realizations.

The multifidelity training dataset consists of the 32-sample HF dataset combined with 116 samples from each of three LF resolutions. These resolutions result from subsampling the 116 HF images using dimensions 32x32, 16x16, and

8x8. This results in a total of $116 \cdot 3 + 32 = 380$ images. For each coarse representation LF_i, $i \in \{1, 2, 3\}$, we add uniform random noise from $\mathcal{U}(0, 0.05 \cdot [\max_{j,k} \text{LF}_i(j,k) - \min_{j,k} \text{LF}_i(j,k)])$. After training, every dataset's accuracy is evaluated on the same validation and test sets described above, i.e., 49 and 35 HF images, respectively. Final accuracy results are reported for predictions on the test set, using the model with the lowest validation loss during training.

C.4.3 Multifidelity loss

The training loss consists of the integral of the Mean Square Error (MSE), assembled from equal contributions (penalty 1/4) of all four fidelities. The integral here is obtained by multiplying each pixel's contribution to the MSE by its size, and it is used to compensate for the different number of pixels present at different fidelities. We also tested a loss formulation where larger penalties were applied to the high-fidelity samples. While this showed some improvements in the final high-fidelity accuracy, the results were not consistently better than with equal penalties across different network weight initializations.

C.4.4 Weight optimization and batch size

Training is performed using the Adam optimizer with a step learning rate scheduler, which decays by a factor 0.9 every s epochs (where s is the LR scheduler step size), with a batch size of 16. Hyperparameters selected are shown in Table 6 and 7.

C.4.5 Weight initialization

We utilize the Xavier initialization scheme [29], where weights are generated using realizations from a normal distribution, with no significant changes in the results.

Acknowledgments

This work was supported by a NSF CAREER award #1942662 (PI DES), a NSF CDS&E award #2104831 (University of Notre Dame PI DES) and used computational resources provided through the Center for Research Computing at the University of Notre Dame. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government. The authors would like to thank Ishani Aniruddha Karmarkar for her assistance with testing fully connected multifidelity network implementations proposed in the literature.

References

[1] M. Dehghani, A. Mehrjou, S. Gouws, J. Kamps, and B. Schölkopf. Fidelity-weighted learning. *arXiv preprint arXiv:1711.02799*, 2017.

[2] S. De, J. Britton, M. Reynolds, R. Skinner, K. Jansen, and A. Doostan. On transfer learning of neural networks using bi-fidelity data for uncertainty propagation. *International Journal for Uncertainty Quantification*, 10(6), 2020.

[3] X. Meng and G.E. Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *Journal of Computational Physics*, 401:109020, 2020.

[4] X. Meng, H. Babaee, and G.E. Karniadakis. Multi-fidelity bayesian neural networks: Algorithms and applications. *Journal of Computational Physics*, 438:110361, 2021.

[5] Yous van Halder, Benjamin Sanderse, and Barry Koren. Multi-level neural networks for PDEs with uncertain parameters. *arXiv preprint arXiv:2004.13128*, 2020.

[6] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[7] S. Minaee, Y.Y. Boykov, F. Porikli, A.J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[8] Lauren Partin, G. Geraci, Ahmad Rushdi, M.S. Eldred, and Daniele Schiavazzi. Multifidelity data fusion in convolutional encoder/decoder assembly networks for computational fluid dynamics applications. In J. Darby

Smith and Edgar Galvan, editors, *Computer Science Research Institute Summer Proceedings 2021*, pages 102–119. The Computer Science Research Institute at Sandia National Laboratories, 2021.

[9] Lauren Partin, Ahmad A. Rushdi, and Daniele E. Schiavazzi. Multifidelity data fusion in convolutional encoder/decoder assembly networks for computational fluid dynamics. In *AIAA SCITECH 2022 Forum*, page 0803, 2022.

[10] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

[11] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*, 2017.

[12] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U.R. Acharya, V. Makarenkov, and S. Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *arXiv preprint arXiv:2011.06225*, 2020.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[14] P. Baldi and P.J. Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26: 2814–2822, 2013.

[15] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[16] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/7edcfb2d8f6a659ef4cd1e6c9b6d7079-Paper.pdf.

[17] D.E. Schiavazzi, A. Nemes, S. Schmitter, and F. Coletti. The effect of velocity filtering in pressure estimation. *Experiments in Fluids*, 58(5):50, 2017.

[18] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[19] D. Xiu and G.E. Karniadakis. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM journal on scientific computing*, 24(2):619–644, 2002.

[20] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4): 541–551, 1989.

[21] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[22] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[23] Fabian Isensee, Paul F Jaeger, Simon AA Kohl, Jens Petersen, and Klaus H Maier-Hein. nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*, 18(2):203–211, 2021.

[24] H. Langseth and L. Portinale. Bayesian networks in reliability. *Reliability Engineering & System Safety*, 92(1): 92–108, 2007.

[25] Carlo Innamorati, Tobias Ritschel, Tim Weyrich, and Niloy J. Mitra. Learning on the edge: Explicit boundary handling in CNNs. *CoRR*, abs/1805.03106, 2018. URL http://arxiv.org/abs/1805.03106.

[26] P. Morerio, J. Cavazza, R. Volpi, R. Vidal, and V. Murino. Curriculum dropout. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3564–3572, 2017. doi: 10.1109/ICCV.2017.383.

[27] A. Gorodetsky, G. Geraci, M.S. Eldred, and J.D. Jakeman. A generalized approximate control variate framework for multifidelity uncertainty quantification. *Journal of Computational Physics*, 408, 2020.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256,

Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.