NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOME COMPARISONS OF NEURAL NETWORK ARCHITECTURES FOR SCIENTIFIC MACHINE LEARNING

by

Javier J. Sustaita

December 2023

Thesis Advisor: Anthony Austin
Second Reader: Wei Kang

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2023	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
SOME COMPARISONS OF NEURAL NETWORK ARCHITECTURES FOR SCIENTIFIC MACHINE LEARNING

5. FUNDING NUMBERS

6. AUTHOR(S) Javier J. Sustaita

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

We compare several neural network architectures for approximating solutions to and solution operators for a handful of elementary 1D partial differential equations. Specifically, we examine whether residual layers offer any benefits over fully connected layers in the context of physics-informed machine learning, finding that the two perform similarly on the problems considered. We also compare the popular DeepONet and Fourier neural operator approaches to operator learning and observe that while the two attain comparable accuracies for linear problems, the latter yields more accurate models in the presence of a simple nonlinearity.

14. SUBJECT TERMS
scientific machine learning, physics-informed neural networks, operator learning, residual neural networks, DeepONet, Fourier neural operator

15. NUMBER OF PAGES
63

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SOME COMPARISONS OF NEURAL NETWORK ARCHITECTURES FOR SCIENTIFIC MACHINE LEARNING

Javier J. Sustaita
Captain, United States Army
BS, United States Military Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
December 2023

Approved by: Anthony Austin
Advisor

Wei Kang
Second Reader

Francis X. Giraldo
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We compare several neural network architectures for approximating solutions to and solution operators for a handful of elementary 1D partial differential equations. Specifically, we examine whether residual layers offer any benefits over fully connected layers in the context of physics-informed machine learning, finding that the two perform similarly on the problems considered. We also compare the popular DeepONet and Fourier neural operator approaches to operator learning and observe that while the two attain comparable accuracies for linear problems, the latter yields more accurate models in the presence of a simple nonlinearity.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

BC	Boundary Condition
DOD	Department of Defense
FFT	Fast Fourier Transform
FNO	Fourier Neural Operator
GELU	Gaussian Error Linear Unit
IC	Initial Condition
NPS	Naval Postgraduate School
PDE	Partial Differential Equation
PINN	Physics-Informed Neural Network

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Science and technology are critical to military success in the 21st century. One emerging technology that has grown in popularity is artificial intelligence, and it seems as though organizations worldwide are trying to determine not *if* artificial intelligence can improve their operations, but *how*. The U.S. military and the Department of Defense (DOD) are no different, at the very least because our adversaries attempt to do the same thing. As General Milley, Army Chief of Staff, put it, "[AI] is a technology our opponents will use against us, and it will be to our benefit to get there first" [1].

The U.S. military and DOD have already begun experimenting with implementing artificial intelligence into military operations. In 2017, the DOD spent $7.4 billion on "big data", artificial intelligence, and the cloud [2] of which $433.6 million went specifically to deep learning and machine learning, which are part of artificial intelligence. Moreover, a recent U.S. Army Research Laboratory report highlights operational applications of machine learning, such as automated target recognition, robotics, cybersecurity, and medical diagnosis, among others [3]. The report also discusses machine learning "research gaps" of military interest. One such gap is the relatively new area of developing "physics-based or physics-like simulations that use machine learning models/equations."

The ability to use machine learning models to simulate physical phenomena will open up even more opportunities for military applications beyond the current focus of automating human tasks. To take one specific example, recent advances in using machine learning to develop reduced-order models for partial differential equations (PDEs) may offer a way to improve weather forecasts. Direct numerical simulations of weather phenomena over large spatial regions are costly due to the high resolution required to resolve clouds. This is dealt with using empirical models called parameterizations to approximate the small-scale physics that drives cloud formation [4]. Tools from the emerging discipline of scientific machine learning are a promising alternative to conventional methods for constructing parameterizations; their development is an active area of research [5]–[8].

In this thesis, we examine the effectiveness of some of these tools by exploring the perfor-

mance of different neural network architectures in approximating solutions to and solution operators for several 1D PDEs. We assess the potential advantages of using neural network models with residual layers in physics-informed machine learning and compare the efficiency of the popular DeepONet and Fourier neural operator (FNO) approaches to operator learning.

Our work provides valuable insights into the effectiveness of various neural network architectures for solving PDEs and learning operators. We find that residual layers may offer some benefits regarding training time but do not consistently yield more accurate approximations than fully-connected layers for the problems we consider. Additionally, our comparison of a modified version of DeepONet and FNO shows that while both methods attain comparable accuracies for linear problems, the latter may offer some benefits in the presence of simple nonlinearities.

The remainder of this thesis is organized as follows. In Chapter 2, we provide the necessary background in machine learning to understand our experiments. Chapter 3 discusses residual networks and their application to PDEs and details our experimental setup and findings. Chapter 4 compares DeepONet and FNO and describes the experiments and results obtained. Finally, Chapter 5 offers a conclusion summarizing our key findings and their implications.

CHAPTER 2:
Background

In this chapter, we introduce and explain the key terms and concepts of machine learning that are necessary for understanding the setup of the experiments conducted in this thesis. Machine learning has gone by many names since its inception in the mid-1900s [9]. Originally thought of as a collection of algorithms that mimic how the human brain thinks logically, machine learning has since become less concerned with artificial intelligence in the classical sense and is now more accurately thought of as a collection of algorithms that enable computers to learn from data [9].

2.1 Neural Networks

Neural networks are a class of machine learning models for approximating essentially arbitrary high-dimensional functions. These networks are built from layers of interconnected nodes known as *neurons*. The most basic type of layer is a *fully-connected layer*. Mathematically, a fully-connected layer is a function of the form

$$\nu_i(x) = \sigma_i(W_i x + b_i). \tag{2.1}$$

Here, the input x is a vector in $\mathbb{R}^n$, W_i is a matrix of *weights*, and b_i is a vector of *biases*. The function σ_i is a nonlinear *activation function*; several examples of common choices for σ_i are shown in Table 2.1. A neural network composed of fully-connected layers is called a *fully-connected neural network* and has the form

$$\nu(x) = (\nu_d \circ \cdots \circ \nu_1)(x). \tag{2.2}$$

The integer d is called the *depth* of the network; it is just the number of layers. The *width* of the network is the number of neurons in each layer and is represented by the number of rows in the W_i matrices. Increasing the depth and width of a network can increase its ability to learn more complex representations of the input data because this increases the number of trainable parameters to fit those relationships; however, increasing the depth and width too much can also result in *overfitting*, which means the network performs well on the training

data but cannot generalize well to new data [10]. This thesis experiments with various depth and width combinations to determine an efficient balance for fitting our models. The width, depth, and all other non-trainable parameters are known as *hyperparameters*.

Table 2.1. Some examples of common neural network activation functions.

Name	Formula
Sigmoid	$1/(1+e^{-x})$
Swish	$x/(1+e^{-x})$
Hyperbolic tangent	$\tanh(x)$
Rectified linear unit (ReLU)	$\max(x, 0)$
Gaussian error linear unit (GeLU)	$\frac{1}{2}x\left(1+\operatorname{erf}\left(x/\sqrt{2}\right)\right)$

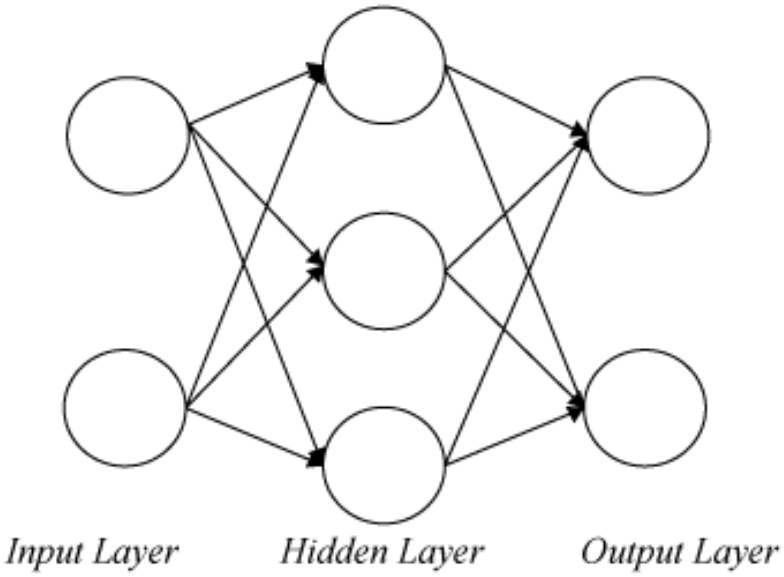

Figure 2.1. A fully-connected network with three layers.

Figure 2.1 depicts a simple three-layer, fully-connected neural network. The circles represent the neurons within each layer, and the arrows represent the connections between the neurons. The network is fully-connected because the neurons within each layer are connected to each neuron of the previous and following layer.

The first layer, or *input layer*, is where the input vector enters the network. The middle layers are known as *hidden layers*; they hold the weights and biases and are where all intermediate calculations are performed. Each neuron in a given layer receives input from each neuron in the previous layer and produces a value sent to each neuron in the subsequent layer. The final layer in the network is the *output layer*; it is where the final calculations are performed, and the overall output is provided [10].

The foundation for using neural networks as general approximators is provided by the so-called universal approximation theorem, the most basic version of which was formulated and proved by Cybenko in 1989 [11]. In the statement of the theorem, a *sigmoidal* function is any real-valued function $\sigma(x)$ that satisfies

$$\sigma(x) \to \begin{cases} 1 & \text{for } x \to \infty \\ 0 & \text{for } x \to -\infty. \end{cases}$$

We denote I_n by the n-dimensional unit cube, $[0,1]^n$, and the space of continuous functions on I_n by $C(I_n)$.

Theorem 2.1 (Universal Approximation Theorem) *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$\nu(x) = \sum_{j=1}^{N} \alpha_j \sigma(W_j^T x + b_j)$$

where $x \in \mathbb{R}$, $W_j \in \mathbb{R}^n$, and $\alpha_j \in \mathbb{R}$, are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum $\nu(x)$ of the above form for which

$$|\nu(x) - f(x)| < \varepsilon$$

for all $x \in I_n$.

This theorem states that a fully-connected neural network with a single hidden layer and a linear output layer can approximate a continuous function to any desired degree of accuracy, provided that the hidden layer is sufficiently wide. Hornik later extended this theorem to apply to networks with non-sigmoidal activation functions, provided the activation functions are sufficiently smooth [12].

One limitation of the universal approximation theorem is that it does not mention the complexity of the neural network required to achieve an approximation of a desired accuracy, nor does it say anything about these networks' ability to overcome the "curse of dimensionality" which distinguishes them from their polynomial and Fourier series counterparts [13]–[15]. The advantage of neural networks over other representations is their ability to efficiently leverage compositional structures, enabling them to approximate complex functions in high-dimensional spaces.

Neural networks can be designed to take on many tasks, of which there are two principal types: classification and regression. A typical example of a classification task is image recognition, in which a model is trained to assign a set of data, such as pixel brightness values, to a category using a probability distribution to identify an object in an image [16]. Regression tasks, on the other hand, predict numerical values, such as using historical weather data to create temperature forecasts. The problems we encounter in this thesis are all of the regression type.

Along with the different types of tasks performed by neural networks, there are also many types of neural network architectures designed to handle specific tasks. These different architectures vary in many ways, including layer types and connections, input and output types, and architecture complexity. In this thesis, we will focus on four different architectures in our experimental comparisons. These are the fully connected network, the residual neural network, and two neural operator architectures known as DeepONet and the Fourier neural operator.

2.2 Training

This section discusses the interworkings of training a neural network through optimization and critical concepts involved in the training process.

2.2.1 Supervised and Unsupervised Learning

Machine learning algorithms can be categorized as either *supervised* or *unsupervised* based on the types of data sets they use [16].

A supervised learning algorithm uses data sets with inputs and their associated outputs. For example, a supervised algorithm may observe an input vector x and its associated output vector y, consequently learning to predict y from x.

An unsupervised learning algorithm uses data sets with inputs without associated outputs. Rather than fitting a specific input-output relationship, unsupervised algorithms attempt to "discover" interesting structures within the input data. For example, given a random vector x, an unsupervised algorithms might try to learn the probability distribution from which x was drawn or the characteristics of that distribution, such as its mean.

The training process introduced in this section will focus primarily on supervised learning algorithms. Unsupervised algorithms are presented in the following section with the introduction of scientific machine learning.

2.2.2 The Loss Function

The *loss function*, which can also be referred to as the *objective* or *cost function*, is used to determine the accuracy of a machine learning model. The process of choosing weights and biases that minimize the loss function is called training the network [10]. There are different types of objective functions based on the tasks the neural network is trying to perform; however, this thesis deals specifically with regression tasks and, therefore, will use an objective function that computes the mean squared error or the average squared difference between the true output values and the output values predicted by the model. Machine learning aims to minimize this loss function to improve the accuracy of the model's predictions.

The prototypical mean squared error loss function in machine learning is defined as follows:

$$L(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} ||f(x_i) - \hat{f}(x_i)||^2 \tag{2.3}$$

where L represents the loss function, f is the true value of the target variable, $\hat{f}$ is the

predicted value of the target variable, N is the number of observations, and the x_i are the training samples.

2.2.3 Stochastic Gradient Descent

In machine learning, minimizing the loss function is done through gradient-based optimization. The classic method for doing this is *gradient descent*. Gradient descent is an iterative optimization algorithm that aims to minimize a function by taking steps in the direction of the steepest descent. Machine learning applies this concept by taking the gradient of the loss function with respect to the model's trainable parameters (i.e., the weights and biases). The downside to this method is that evaluating the loss function at each point in the training set for every iteration is computationally expensive [10].

For this reason, many machine learning algorithms employ a cheaper alternative known as *stochastic gradient descent*. Stochastic gradient descent differs from ordinary descent in that it takes the gradient at a randomly selected subset of training points known as a *minibatch* rather than all of them to reduce computational costs. This reduction is partly due to the reduction in memory required to process the smaller amounts of data rather than the entire sample. The algorithm selects random mini-batches "without replacement" until all training points have been selected. A full pass over the training data is known as an *epoch*. Finally, the size of each iteration step is known as the *learning rate* of the algorithm and is a "key ingredient" in the performance of these computations [10].

In the experiments conducted for this thesis, we chose an optimization algorithm with an adaptive learning rate called *Adam* [17]. Adam is a variant of stochastic gradient descent that uses both the first and second moments of the gradients to update the model parameters and includes a bias correction step to account for their initialization at the origin [16, Section 8.5.3].

2.2.4 Automatic Differentiation

Computing the gradients of the loss functions is necessary for machine learning optimization, but this can be challenging when done manually due to the complexity of most machine learning models. For this reason, *automatic differentiation* is used instead.

Automatic differentiation "refers to a specific family of techniques that compute derivatives by accumulating values during code execution to generate numerical derivative evaluations rather than derivative expressions. This allows accurate evaluation of derivatives at machine precision with only a small constant factor of overhead and ideal asymptotic efficiency" [18]. Specifically, machine learning uses *reverse mode* automatic differentiation, in which gradient information is accumulated "backward" from the loss to the input, a process commonly known as *backpropagation* [16].

This thesis used *TensorFlow*, an open-source end-to-end machine learning platform created by Google, to build and train our models. Packages and tools integrated into TensorFlow simplify the automatic differentiation process for training machine learning models [19].

2.3 Physics-Informed Neural Networks (PINNs)

Scientific machine learning, which applies machine learning to scientific computing, aims to solve complex scientific problems by integrating domain-specific knowledge and constraints into machine learning models. This section introduces an approach to scientific machine learning called physics-informed neural networks (PINNs) [20]. PINNs aim to incorporate physics-based constraints into the neural network training to enable accurate and efficient learning from limited data. PINNs have shown promising results in various applications, such as fluid dynamics, electromagnetics, and mechanics [21].

The primary difference between standard fully-connected networks and PINNs is simply the construction of the loss function. In addition to the typical loss function that aims to minimize the difference between actual and predicted outputs, PINNs leverage physics-based constraints by including them in the loss function. These constraints are added by minimizing three components of the differential equation being solved: the residual of the differential equation, the error in the boundary condition fit, and the error in the initial condition fit [20].

In more detail, consider the following initial boundary-value problem

$$\begin{aligned} &\mathcal{N}[u] = 0 && \text{for } u(x,t), \quad x \in \Omega, \quad t \in [0,T] \\ &u(x,t) = a(x) && \text{for } x \in \partial X \\ &u(x,0) = g(x) \end{aligned}$$

where $u(x,t)$ is the solution to the problem, $\mathcal{N}$ is a potentially nonlinear differential operator, and Ω is a subset of $\mathbb{R}^D$. We aim to build a PINN $\hat{u}$ that approximates u. According to [20], $\hat{u}(x,t)$ can be learned by minimizing the mean squared error loss

$$L(\hat{u}) = L_{IC}(\hat{u}) + L_{BC}(\hat{u}) + L_{PINN}(\hat{u}) \tag{2.4}$$

where

$$L_{IC}(\hat{u}) = \sum_{i=1}^{N_{IC}} \left|\hat{u}(x_i^{IC}, 0) - g(x_i^{IC})\right|^2, \tag{2.5}$$

$$L_{BC}(\hat{u}) = \sum_{i=1}^{N_{BC}} \left|\hat{u}(x_i^{BC}, t_i^{BC}) - a(x_i^{BC})\right|^2, \tag{2.6}$$

and

$$L_{PINN}(\hat{u}) = \sum_{i=1}^{N_{PINN}} |\mathcal{N}(\hat{u}(x_i, t_i))|^2. \tag{2.7}$$

The $L_{IC}(\hat{u})$ and $L_{BC}(\hat{u})$ terms penalize the mismatch in $\hat{u}$ at the initial condition and boundary condition training points $(x_i^{IC}, 0)$ and (x_i^{BC}, t_i^{BC}), respectively, while the $L_{PINN}(\hat{u})$ term enforces adherence of the model to the physics described by the differential equation at a finite set of collocation points (x_i, t_i) [20]. The $L_{PINN}(\hat{u})$ term is an example of an unsupervised loss, which differs from the mean-squared error loss described in the previous section.

This combination of terms is a crucial component of PINNs as it enforces an accurate fitting to the training data and the physical laws described in the PDEs.

CHAPTER 3:
Residual Layers in Physics-Informed Learning

In this chapter, we introduce networks with residual layers and describe some experiments comparing PINNs built with this architecture with ones built from fully-connected networks. Our goal is to determine whether using residual layers instead of fully-connected layers in PINNs provides either improved accuracy or reduced training time.

3.1 Residual Learning

Residual layers are a type of neural network layer that introduces "shortcut connections" between a hidden layer's inputs and its outputs. This output is known as a "residual function," which outputs the difference between the layer's output and input. The purported benefit of adding these connections is eliminating training loss errors in neural networks with greater depth [22].

Recall the formula for a layer in a fully-connected network,

$$v(x) = \sigma(Wx + b).$$

A residual network consists of layers of the form

$$\rho(x) = \sigma\big(x + W_2\sigma(W_1x + b_1) + b_2\big),$$

where W_1 and W_2 are weights, b_1 and b_2 are biases, and σ is a nonlinear activation function. These residual layers act as a correction or adjustment to the input data in x. An example of a residual layer can be found in Figure 3.1.

3.2 Experimental Setup

To investigate whether using residual layers provides any benefits, we performed a set of experiments in which we compared PINNs built from fully-connected layers with ones built from residual layers for a pair of elementary PDEs. Specifically, we considered the uniform

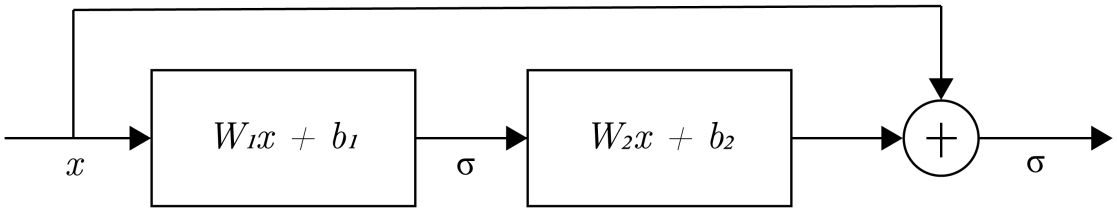

Figure 3.1. Residual learning: a building block. Source: [22, Figure 2].

heat equation in 1D with thermal diffusivity k,

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial^2 x} = 0, \tag{3.1}$$

and the linear advection equation in 1D with wave speed c,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0. \tag{3.2}$$

In both cases, the equations are posed for $-1 \leq x \leq 1$ and $0 \leq t \leq 1$ with periodic boundary conditions $u(-1,t) = u(1,t)$. We consider two initial conditions: $u(x,0) = u_0^1(x) = \sin(2\pi x)$, in which case we have

$$u(x,t) = \begin{cases} e^{-k(2\pi)^2 t}\sin(2\pi x) & \text{for (3.1)} \\ \sin\big(2\pi(x-ct)\big) & \text{for (3.2),} \end{cases}$$

and $u(x,0) = u_0^2(x) = \sin(2\pi x) + \sin(5\pi x)$, for which

$$u(x,t) = \begin{cases} e^{-k(2\pi)^2 t}\sin(2\pi x) + e^{-k(5\pi)^2 t}\sin(5\pi x) & \text{for (3.1)} \\ \sin\big(2\pi(x-ct)\big) + \sin\big(5\pi(x-ct)\big) & \text{for (3.2).} \end{cases}$$

3.3 Experiments

3.3.1 Models

We trained various PINN models for (3.1) and (3.2) based on fully-connected and residual architectures. For the fully-connected models, we used networks of depths 8, 12, and 16. For the residual networks, we used depths of 4, 6, and 8 to yield models with the same number of trainable parameters as the fully-connected ones. Table 3.1 lists the numbers of trainable parameters for each depth. All layers in both types of networks were taken to have a width of 20. For the activation function σ, we used the hyperbolic tangent (see Table 2.1).

Table 3.1. Network depths and trainable parameters (all widths set to 20).

FNN Layers	Residual Layers	Trainable Parameters
8	4	3,441
12	6	5,121
16	8	6,801

3.3.2 Training Data

The training data for the PINNs was generated by selecting sets of points x_i^{IC} in space and times t_i^{BC} at which to penalize the model error in satisfying the initial conditions (ICs) and boundary conditions (BCs), respectively. The 200 IC training points were created by defining a spatial domain and generating equally spaced spatial coordinates. The 100 BC training points were generated by assigning specific values to the spatial coordinates at the boundaries while varying the time coordinate. The 8000 additional training points (x_i, t_i) required for the PINN were generated using the Latin hypercube sampling method [23].

3.3.3 Optimizer

The Adam optimizer with a learning rate of 10^{-3} was chosen for these experiments. All networks were trained using a full-batch method to process all data points simultaneously.

3.3.4 PINN Loss Function

The loss functions for the PINNs are similar to equation (2.4) and consist of three main components: the PDE residual term, the IC mismatch term, and the BC enforcement term.

The PDE residual term penalizes the deviation of the network-predicted solution from satisfying the PDE. The code generated for these experiments takes the gradients of the network's predicted solution for the spatial and temporal coordinates. Then it evaluates the PDE residual equation for heat and advection in 1D as follows:

$$L_{PINN}(\hat{u}) = \begin{cases} \sum_{i=1}^{N_{PINN}} |\hat{u}_t(x_i, t_i) - k\hat{u}_{xx}(x_i, t_i)|^2 \text{ for (3.1)} \\ \sum_{i=1}^{N_{PINN}} |\hat{u}_t(x_i, t_i) + c\hat{u}_x(x_i, t_i)|^2 \text{ for (3.2).} \end{cases}$$

where $\hat{u}$ is our PINN approximation of the solution u, the wave speed in the advection equation is $c = 1.0$, and the diffusivity in the heat equations is $k = 0.075$ for these experiments.

The IC mismatch term penalizes the discrepancy between the network's predicted solution at the initial time and the known initial condition by computing the mean square difference between the predicted and actual ICs. The IC equation has the same form for both PDEs.

$$L_{IC}(\hat{u}) = \sum_{i=1}^{N_{IC}} |\hat{u}(x_i^{IC}, 0) - u_0(x_i^{IC})|^2.$$

The BC enforcement term ensures the network's predictions satisfy the given BCs by evaluating the discrepancy between the predicted solutions and the left and right boundaries. Similar to the IC equation, the BC form is the same for both PDEs.

$$L_{BC}(\hat{u}) = \sum_{i=1}^{N_{BC}} |\hat{u}(1, t_i^{BC}) - \hat{u}(-1, t_i^{BC})|^2.$$

These three terms are combined by taking their sum and used as the loss function during the training process. By minimizing this loss function, the PINN model is able to learn solutions that satisfy the PDE, IC, and BC constraints. The combining formula has the same

form as seen in Chapter 2.

$$L(\hat{u}) = L_{IC}(\hat{u}) + L_{BC}(\hat{u}) + L_{PINN}(\hat{u}).$$

3.4 Findings and Results

The findings for these experiments are broken down into two categories: training time and model accuracy.

In these experiments, the models were trained out to 5000 epochs, and models were saved at every 500 epochs to determine how much training duration affected the models. We also saved the models that yielded the minimum values of the losses. After training the models for each network structure, the training loss for each PINN was plotted for all network structures to approximate the training time for each configuration. The PINN approximations were plotted against manufactured exact solutions at two separate time steps ($t = 0$ and $t = 1$) to determine which network structures, if any, provide advantages in accuracy.

3.4.1 Training Time

To compare the training time between the models, we plot the training loss for each architecture as a function of the training epoch.[1] The plots, such as those shown in Figure 3.2, display a "spiky" behavior in the training loss, which can result from several factors. This is a known behavior of the Adam optimizer [24]; however, since the training loss for our PINN models consistently portrayed a downward trend, we generated plots that capture the lower envelope of the loss values for each architecture to make the comparison. This adaptation can be seen in Figure 3.2, where the behavior of the loss plots is accompanied by a line representing the lower envelope of the values.

[1] We measure the training time in terms of epochs, not wall time.

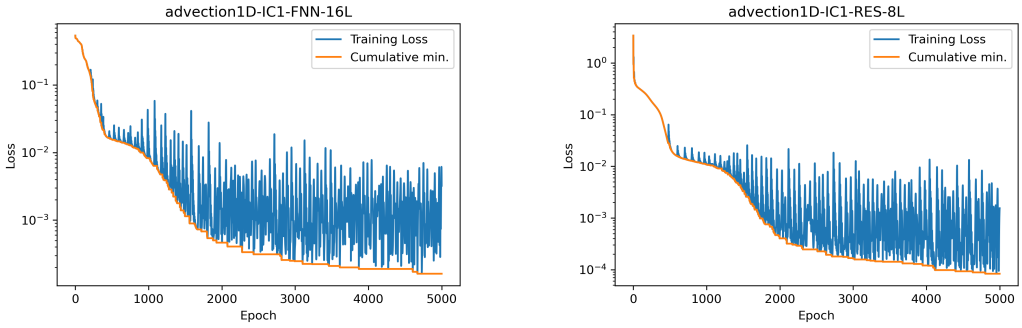

Figure 3.2. Typical training loss plots for a fully-connected 16-layer network (left) and an 8-layer network with residual layers (right).

The left side of Figure 3.3 compares the two architectures for the 1D advection equation trained on the first initial condition. Both network architectures train at the same speed except for the 16-layer fully-connected network, which does not achieve as low a loss as the other models at 5000 epochs. The training loss for both architectures appears to plateau at approximately 10^{-4} given the same training time showing that neither architecture provides an advantage over the other.

The right side of Figure 3.3 makes the same comparison for the second, more complex IC. Neither architecture reaches as low a loss value as before. Still, the 12-layer fully-connected network avoided plateauing with the other five models and appears to continue downward at the 5000 epoch mark. This trend suggests that given enough training time, this model could achieve similar losses seen for the first IC.

Figure 3.4 continues the architecture comparisons by replacing the advection equation with the 1D heat equation. Given the first IC (left side), both architectures show similar results, with the networks with residual layers attaining slightly lower losses overall at 5000 epochs. Despite the promising results with the first IC, introducing the second IC produces similar degraded results as seen in the advection models. The right side of Figure 3.4 shows that given a more complex initial condition, both the fully-connected networks and the networks with residual layers take longer to train, with the residual networks training slightly faster.

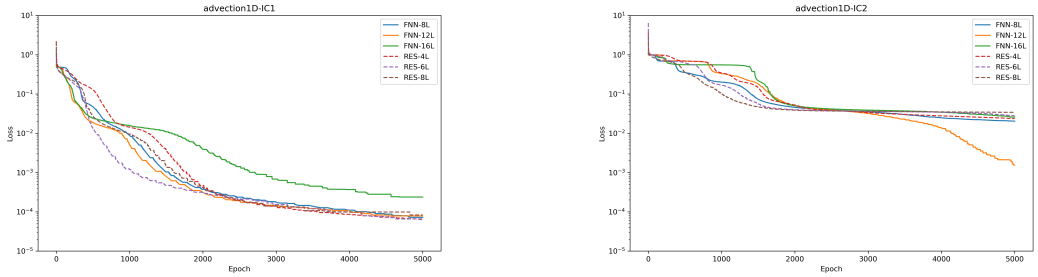

Figure 3.3. Lower envelope of training loss for the fully-connected networks (solid) and the networks with residual layers (dashed) for the 1D advection equation for IC1 (left) and IC2 (right).

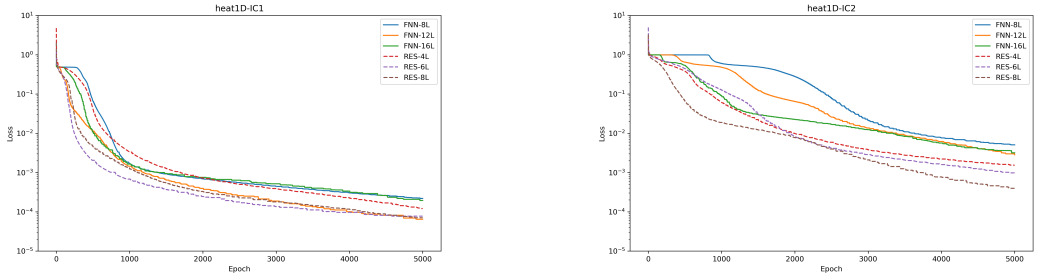

Figure 3.4. Lower envelope of training loss for the fully-connected networks (solid) and the networks with residual layers (dashed) for the 1D heat equation and IC1 (left) and IC2 (right).

3.4.2 Accuracy

The next step in these experiments was to see how the models' training loss translated into the PINNs' accuracy compared to the PDEs' exact solutions.

We assess the accuracy of the models qualitatively by plotting the PINN solutions from the models attaining the minimum loss values against the know exact solutions of the PDEs. The four figures generated for these experiments compare all network depths for both architectures using the same two PDEs and two ICs at the initial and final times.

As shown in Figure 3.5, for the 1D advection equation given the first IC, both network structures produced accurate PINN approximations at all depths, with neither architecture

performing better. These results confirm our interpretation of the loss values achieved by the models.

Figure 3.6 shows the results for the second IC, where the degraded loss values shown in the previous section can be seen in the degraded solution plots. Neither architecture was as accurate as when trained on the first IC, and all models appeared to suffer accuracy within the same spatial intervals at both the initial and final time stamps.

For the 1D heat equation given the first IC, both architectures struggled to accurately match the exact solution to the final training time, especially near the interval endpoints. As could be predicted from the observed training losses, introducing the second IC resulted in even less accurate models. Both network structures produced inaccurate approximations, with neither architecture producing noticeable advantages (see Figure 3.8).

Despite networks with residual layers appearing to train faster on a slightly more consistent basis, the PINN approximations did not exhibit a significant advantage over the fully-connected networks. The fully-connected networks produced similar approximations to those with residual layers despite not always reaching the same training loss. We conjecture that training the models longer on the more complex IC will continue to produce more accurate approximations, with neither architecture gaining a significant advantage.

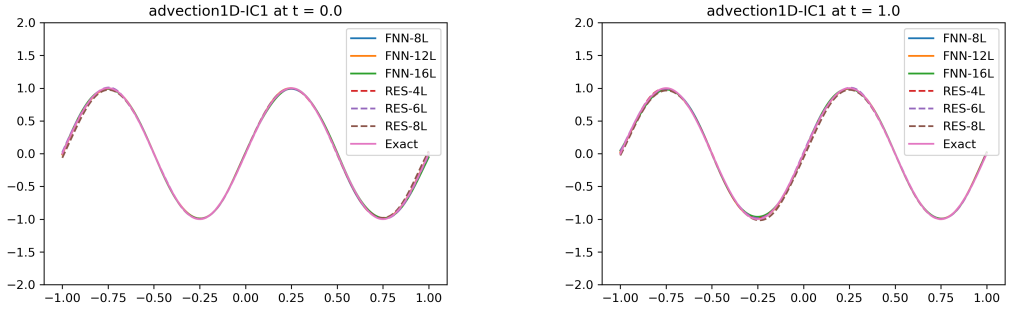

Figure 3.5. 1D advection solution plots for all networks at initial (left) and final (right) times given IC1.

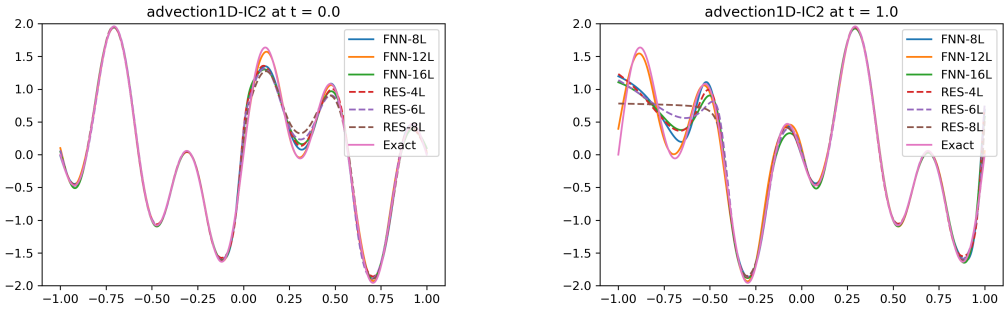

Figure 3.6. 1D advection solution plots for all networks at initial (left) and final (right) times given IC2.

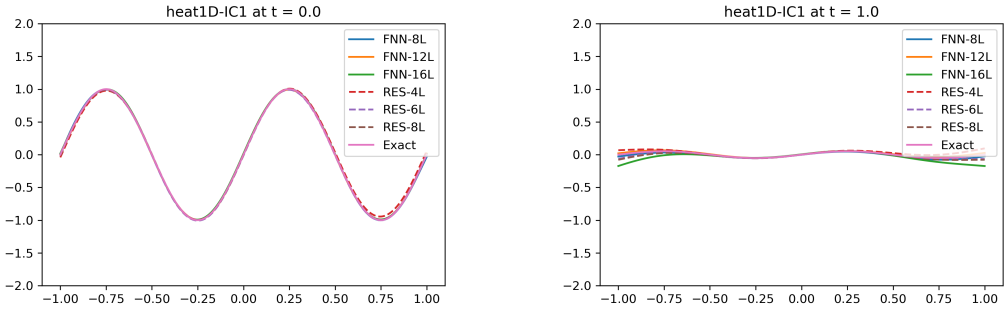

Figure 3.7. 1D heat solution plots for all networks at initial (left) and final (right) times given IC1.

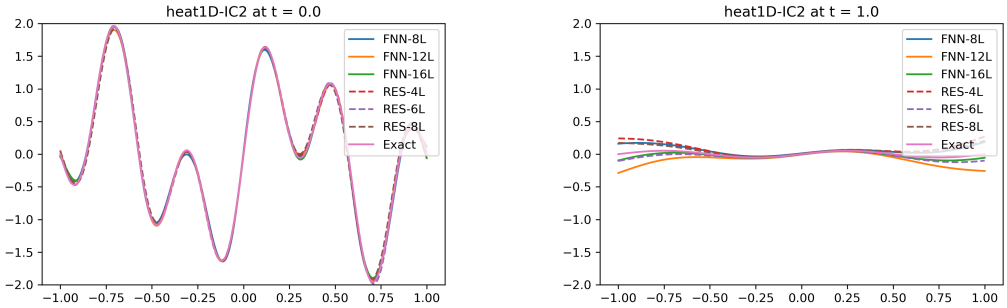

Figure 3.8. 1D heat solution plots for all networks at initial (left) and final (right)l times given IC2.

3.4.3 Variance With Initial Guess

Machine learning algorithms typically initialize the model parameters with random values at the start of training. In our case, the model's weights were initialized using the default Glorot uniform initializer in Tensorflow [25], while the biases were initialized to zero. To understand the effect of this randomness on our results, we ran the experiments three times using the same setup to avoid basing our evaluations on single-run outliers.

Figures 3.9, 3.10, and 3.11 provide examples of loss values and resultant solution plots over three training runs for the 1D advection equation with the second IC. Despite the varying results produced by the randomness of the initialization, no architecture displayed a significant advantage over the three training runs, and all of them struggled to match the exact solution within the same set of points. For example, in the first two runs, the fully-connected 12-layer model achieves the lowest loss and appears to produce slightly more accurate approximations (see Figures 3.9 and 3.10); however, in the third run, the networks with residual layers achieved lower loss values and also marginally better approximations (see Figure 3.11). Based on these observations, the variance between training runs does not pose a significant issue in our final results and evaluations of the models.

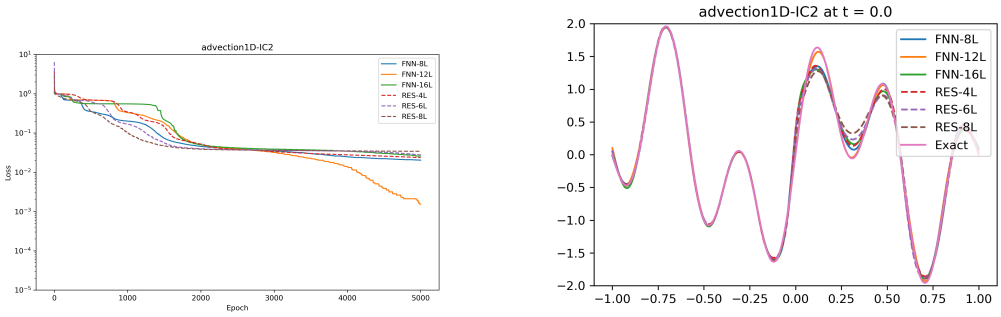

Figure 3.9. The first run training loss for the 1D advection equation for IC2 (left) and resultant solution plot (right).

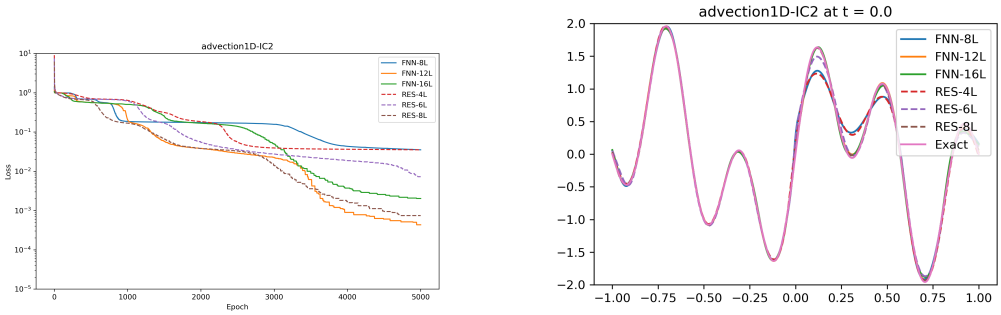

Figure 3.10. The second run training loss for the 1D advection equation for IC2 (left) and resultant solution plot (right).

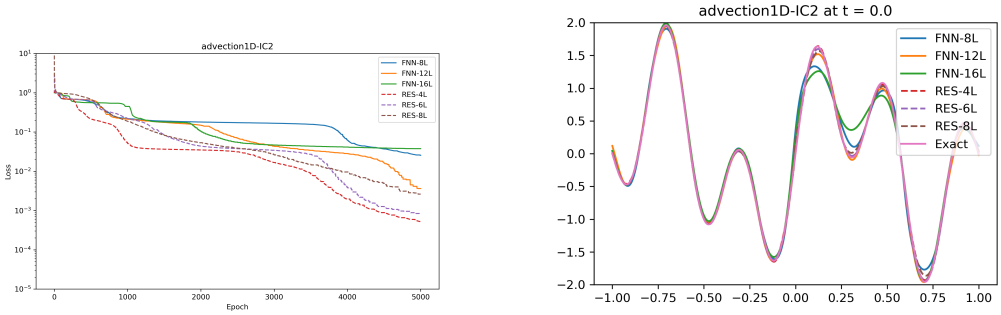

Figure 3.11. The third run training loss for the 1D advection equation for IC2 (left) and resultant solution plot (right).

3.5 Conclusions

Residual networks appear to give marginal advantages in training time but do not necessarily produce more accurate approximations at the current training parameters. Upon training all models with identical parameters, the lowest loss values for these experiments are on the order of 10^{-4}.

CHAPTER 4:
A Comparison of Two Neural Operators

In the previous chapter, we looked at constructing neural network approximations to the solution to a single initial-boundary value problem i.e., we aimed to approximate the solution $(x, t) \mapsto u(x, t)$ to a PDE for a single given initial condition u_0. This chapter aims to construct neural network approximations to solve *any* initial-boundary value problem for the PDE, i.e., we aim to approximate the solution *operator* $(u_0, x, t) \mapsto u(x, T)$. We consider two emerging neural operator architectures: deep operator networks (DeepONets) and Fourier neural operators (FNOs).

The work in [26] evaluated the performance of these two neural operators against one another for simulating complex dynamics and system identification. They concluded that the two operators are comparable for relatively simple settings but that DeepONet outperforms FNO in the presence of complex domain geometries.

This chapter details our experiments comparing the two operators to determine whether DeepONet or FNO can offer improved performance by conducting experiments similar to those seen in Chapter 3. Before describing the experiments comparing the two operators, we will briefly overview their characteristics.

4.1 Operator Learning Problem

The basis for the DeepONet and FNO architectures is the operator learning problem which can be mathematically represented as

> Find G such that $G(u_0) = u$ for all ICs u_0, where u is the solution to $\mathcal{N}[u] = 0$, $u(x, 0) = u_0(x)$, subject to some BCs.

The problem above aims to find a solution operator, G, for the PDE that maps the input u_0 to output u. The input u_0 is discretized by sampling it at a finite number of locations $x_1, x_2, \ldots, x_n$ [27]. Figure 4.1 shows a diagram of a general operator network.

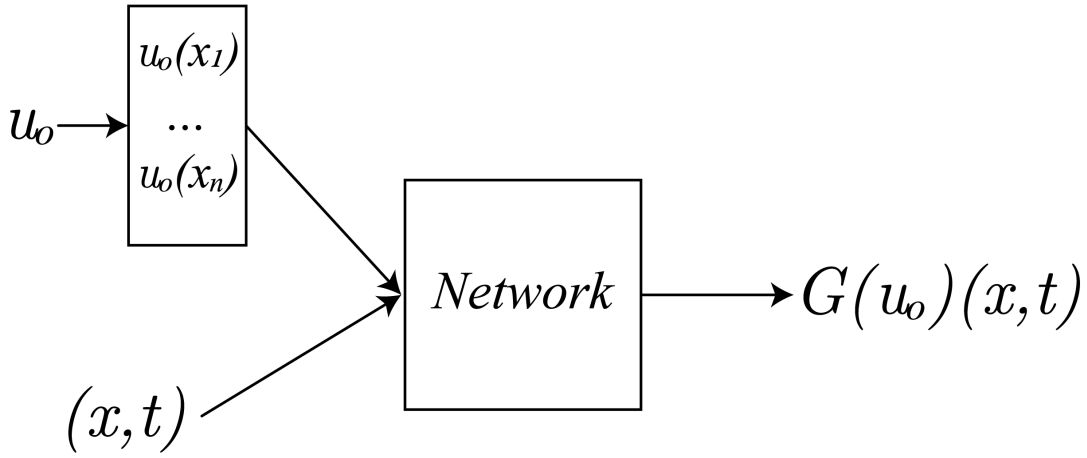

Figure 4.1. Network to learn operator G.

Due to some of the challenges of learning complex operators, researchers have experimented with different architectures to minimize optimization and generalization errors. The first of two emerging approaches to these experiments we cover is DeepONets.

4.1.1 DeepONets

DeepONets can be described as a network consisting of two sub-networks [26]. The first sub-network is the "trunk" network that takes in (x, t) and outputs $[\varphi_1, \varphi_2, ..., \varphi_p]^T \in \mathbb{R}^p$. The second sub-network is made up of p "branch" networks that each take $[u_0(x_1), u_0(x_2), ..., u_0(x_n)]^T$ as the input and together output $[\alpha_1, \alpha_2, ..., \alpha_p]^T \in \mathbb{R}^p$. These two sub-networks are then merged to produce the approximation:

$$G(u_0)(x,t) \approx \sum_{k=1}^{p} \alpha_k(u_0)\varphi_k(x,t), \tag{4.1}$$

The right-hand side of (4.1) takes the same general form as the linear basis expansion models commonly used in applied mathematics, except that the basis functions are learned as neural networks, and the coefficients also come from neural networks, e.g., orthogonal projection.

The branch and trunk networks can have any architecture, including those with fully-

connected and residual layers. The type of network selected is typically based on the structure of the input function u_0 [26]. The general setup of a DeepONet model can be seen in Figure 4.2.

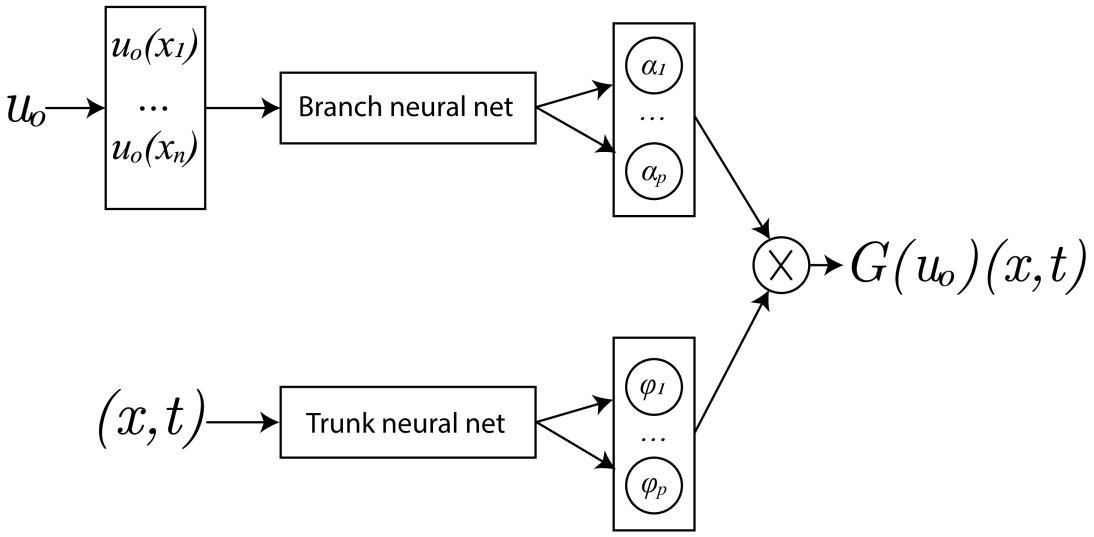

Figure 4.2. DeepONet Architecture.

4.1.2 Fourier Neural Operators

The second emerging approach that addresses the operator learning problem is Fourier neural operators, which attempt to leverage Fourier analysis for efficient approximations.

An FNO is a neural network of Fourier layers that acts as a low-pass filter on the input data. The primary network parameters, i.e., the filter coefficients, are defined and learned in Fourier space [26].

The FNO architecture starts by first lifting the function value $u(x)$ to a higher dimensional representation of the form

$$z_0(x) = P(u(x)) \in \mathbb{R}^{d_z}$$

for any location x on the mesh. Here, P is a local transformation parameterized by a shallow fully-connected neural network. It then successively applies L Fourier layers to z_0.

The Fourier layers start from an input $z(x)$ and apply the Fourier transform $\mathcal{F}$. This is

followed by a linear transform R, the trainable portion of the layer, on the lower Fourier modes. Modes higher than a preselected *cutoff* are filtered out. The layers then apply the inverse Fourier transform $\mathcal{F}^{-1}$ to the filtered data. Alongside the Fourier transforms in each Fourier layer, a matrix multiplication $W_\ell \cdot z_\ell$ is added to the output through a residual-like connection where W_ℓ is a weight matrix. The layer concludes by applying an activation. This process is depicted in the bottom portion of Figure 4.3. The output of the $(\ell + 1)$th Fourier layer $z_{\ell+1}$ has the form

$$z_{\ell+1} = \sigma\big(\mathcal{F}^{-1}\big(R_\ell \cdot \mathcal{F}(z_\ell)\big) + W_\ell \cdot z_\ell + b_\ell\big),$$

where σ is a nonlinear activation function and b_ℓ is a bias vector.

The output of the last Fourier layer, z_L, then goes through another local transformation Q resulting in an output function of the form

$$G(u) = Q(z_L(x)).$$

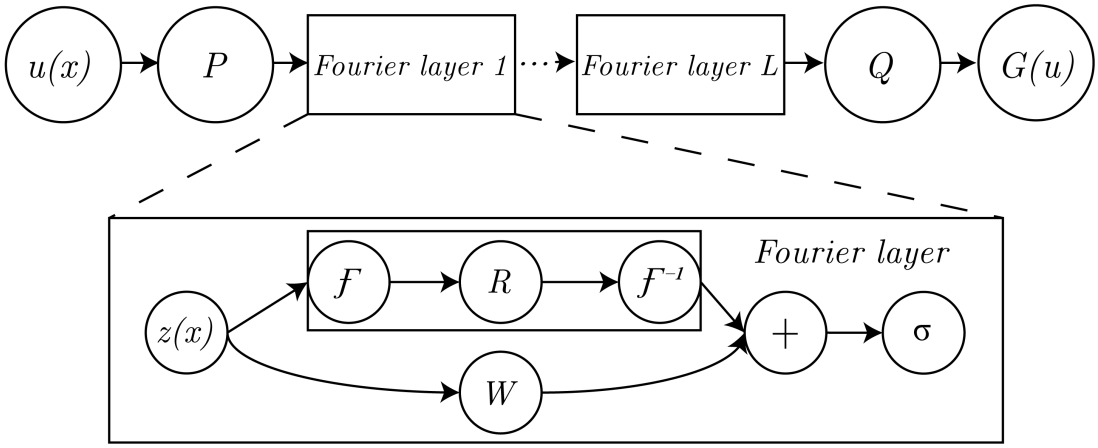

Figure 4.3. FNO Architecture (top) and Fourier layer breakdown (bottom). Source: [26, Figure 3].

4.2 Experimental Setup

To investigate which neural operator architecture would offer better performance, we performed simple experiments comparing DeepONets and FNOs for three elementary PDEs. Specifically, we considered the uniform heat equation in 1D with thermal diffusivity $k = .05$,

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} = 0, \tag{4.2}$$

the linear advection equation in 1D with wave speed $c = 1.0$,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \tag{4.3}$$

and the Burgers equation in 1D with viscosity $\nu = 10^{-2}$,

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{4.4}$$

4.3 Experiments

4.3.1 Models

We trained a variety of DeepONet and FNO models to approximate the solution operators for each of these PDEs.

One significant difference between DeepONets and FNOs is the latter maps the discretization of u_0 in an equispaced mesh to the discretization of $G(u_0)$ on the same mesh.[2] Based on this difference, we modified our DeepONet models to work with a fixed equispaced grid to compare both neural operators on the same training data and to ensure both neural operators approximated the same mapping. Specifically, DeepONets as presented in [27] implement a map of the form

$$u_0(x_1), \ldots, u_0(x_k), x, t \mapsto u(x, t), \tag{4.5}$$

[2]This is done to facilitate efficient implementation of the Fourier layers via the fast Fourier transform (FFT).

while the map implemented by an FNO is

$$u_0(x_1), \ldots, u_0(x_k) \mapsto u(x_1, T), \ldots, u(x_k, T). \tag{4.6}$$

We discretized the output variable of our DeepONets so that they also implement the FNO map (4.6).

For the DeepONet models, the three PDEs were used to train networks of widths 64, 128, and 256, all at depths 4, 5, and 6. These nine configurations for each PDE were evaluated at 500, 1000, 1500, 2500, and 5000 training points for 108 models.

For the FNO models, the three PDEs were used to train networks of widths 64, 96, and 128, all at depths 4, 5, and 6. In addition to training these nine configurations for each PDE, they were evaluated at 8, 12, and 16 cutoff values for 324 models. The dimensions of the models for both architectures, along with their respective number of trainable parameters, can be seen in Tables 4.1 and 4.2.

Table 4.1. DeepONet depths, widths, and trainable parameters.

DeepONet		
Depth	Width	Parameters
4	64	90,688
4	128	230,528
4	256	657,664
5	64	99,008
5	128	263,552
5	256	789,248
6	64	107,328
6	128	296,576
6	256	920,832

Table 4.2. FNO depths, widths, and trainable parameters.

Fourier Neural Operator			
Depth	Width	Cutoff	Parameters
4	64	8	156,353
4	64	12	221,889
4	64	16	287,425
4	96	8	351,265
4	96	12	498,721
4	96	16	646,177
4	128	8	624,001
4	128	12	886,145
4	128	16	1,148,289
5	64	8	198,281
5	64	12	275,201
5	64	16	357,121
5	96	8	434,305
5	96	12	618,625
5	96	16	802,945
5	128	8	771,585
5	128	12	1,099,265
5	128	16	1,426,945
6	64	8	230,209
6	64	12	328,513
6	64	16	426,817
6	96	8	517,345
6	96	12	738,529
6	96	16	959,713
6	128	8	919,169
6	128	12	1,312,385
6	128	16	1,705,601

4.3.2 Training Data

We trained the models using a supervised learning scheme with a standard mean-squared error loss, as described in section 2.2.

For training data, we used the Chebfun software package in MATLAB [28] to generate 5000 random smooth functions [29] to serve as input initial conditions.

The corresponding outputs were generated using the exact solution in the case of 1D advection, and high-accuracy approximations were computed using exponential integrators in the cases of 1D heat and Burgers. The calculations for the latter were performed with the aid of the "spin" function in Chebfun [30].

In addition to the 5000 training points used to train the models, we also generated 200 validation points. This validation data was not a part of the optimization process, meaning it did not affect adjusting weights and biases but was used to assess the performance of the neural operator networks on unseen data [10].

4.3.3 Hyperparameters

The Adam optimizer with an initial learning rate of 10^{-3} was chosen for these experiments. We used an exponential decay learning rate scheduler that decreased the learning rate by 0.5 every 20,000 epochs for the DeepONet models and every 1000 epochs for the FNO models. All networks were trained using a batch size of 500, and the activation function used for both the DeepONet and FNO models was the Gaussian Error Linear Unit (GELU) (see Table 2.1).

4.4 Findings and Results

The findings for these experiments are broken into two categories: validation loss and solution accuracy.

In these experiments, the DeepONet models were trained to 200,000 epochs, and the FNO models were trained to 10,000 epochs.

4.4.1 Validation Loss

Figures 4.4-4.7 show tables containing the training and validation loss for all of the DeepONet models, and Figures 4.8-4.11 show the tables for all of the FNO models. The cells are color formatted to show losses in green belonging to the best-performing networks, and the losses in red belonging to the worst-performing networks. We determined that any loss below 10^{-5} would be considered effectively optimal, and any loss above 10^{-2} would be considered poor performance.

Based on these conditions, we can see in Figure 4.4 that the DeepONet models achieved an optimal training loss for the 1D advection and heat equations and a suboptimal loss for the 1D Burgers equation for all network configurations at the lowest data point setting. Despite the optimal training losses achieved, the validation losses across the board were suboptimal for all network configurations given 500 training points. The effect of the amount of data used to train the models becomes very apparent by looking at the other DeepONet loss tables. As training data increases, the associated validation losses decrease even to optimal values except for the 1D Burgers equation. Even given our maximum of 5000 training points, the DeepONet models could not achieve optimal training or validation losses for Burgers.

The FNO tables show a similar picture given low training points, albeit with marginally better loss values for the 1D Burgers equation. However, the FNO models given 5000 training points performed significantly better by reaching optimal training loss values given the more complex PDE. Although the FNO models still do not achieve values below 10^{-5} for Burgers, we believe that adding more training data will improve the models, eventually reaching optimal values.

The takeaway from our experiments is that FNO models achieved lower training and validation loss values than the DeepONet models when given greater training data. Although the DeepONet models achieved optimal numbers with the less complex PDEs, increasing the training data for these models did not benefit them as significantly as with the FNOs.

			Width					
			64		128		256	
			Val Loss	Final Loss	Val Loss	Final Loss	Val Loss	Final Loss
Depth	4	advection1D	1.35E-03	4.18E-07	1.25E-03	3.57E-07	3.09E-03	3.39E-06
		burgers1D	8.23E-01	2.07E-03	4.28E-01	8.30E-05	3.60E+01	1.06E-03
		heat1D	3.42E-04	1.20E-07	7.72E-04	1.05E-07	2.71E-04	8.69E-08
	5	advection1D	5.41E-03	9.89E-08	8.12E-03	7.55E-08	9.12E-03	9.29E-08
		burgers1D	8.01E-01	5.96E-04	2.67E-01	2.55E-05	3.29E-01	3.56E-01
		heat1D	6.37E-04	5.64E-08	1.40E-03	2.26E-08	2.36E-03	1.36E-08
	6	advection1D	7.47E-03	4.38E-08	2.79E-02	1.84E-08	1.21E-02	2.54E-08
		burgers1D	4.40E-01	2.44E-04	1.30E-01	7.26E-06	3.29E-01	3.56E-01
		heat1D	1.34E-03	4.85E-09	4.97E-03	3.00E-09	3.41E-03	5.05E-09
T = 500								

Figure 4.4. Loss values for DeepONet models at 500 training points.

			Width					
			64		128		256	
			Val Loss	Final Loss	Val Loss	Final Loss	Val Loss	Final Loss
Depth	4	advection1D	9.39E-05	2.03E-07	7.77E-04	4.45E-08	1.17E-05	6.32E-08
		burgers1D	5.54E-02	2.13E-03	9.75E-02	1.32E-04	5.09E-01	1.60E-04
		heat1D	3.16E-05	3.45E-08	9.13E-05	3.56E-09	4.68E-06	4.00E-08
	5	advection1D	1.08E-03	1.41E-08	1.44E-03	1.87E-08	2.30E-03	4.66E-08
		burgers1D	9.80E-02	6.11E-04	7.46E-02	2.64E-05	9.81E-02	1.22E-05
		heat1D	1.34E-04	8.68E-09	3.14E-04	1.08E-08	1.26E-04	1.84E-08
	6	advection1D	1.84E-03	1.72E-08	4.53E-03	7.04E-09	6.12E-03	9.35E-09
		burgers1D	9.81E-02	2.75E-04	6.17E-02	8.88E-06	3.29E-01	3.45E-01
		heat1D	2.10E-04	2.63E-09	4.88E-04	2.40E-09	7.36E-04	4.93E-09
T = 1000								

Figure 4.5. Loss values for DeepONet models at 1000 training points.

			Width					
			64		128		256	
			Val Loss	Final Loss	Val Loss	Final Loss	Val Loss	Final Loss
Depth	4	advection1D	3.83E-08	1.49E-08	3.60E-07	1.83E-08	1.04E-07	3.57E-08
		burgers1D	9.96E-03	1.61E-03	2.06E-02	1.36E-04	3.29E-01	3.41E-01
		heat1D	9.44E-09	3.55E-09	1.78E-08	3.89E-09	4.49E-08	1.28E-08
	5	advection1D	3.44E-05	4.55E-07	1.47E-04	5.35E-09	2.56E-07	2.01E-08
		burgers1D	1.29E-02	4.77E-04	1.76E-02	2.91E-05	3.29E-01	3.41E-01
		heat1D	3.56E-06	5.09E-08	2.21E-05	1.42E-09	4.71E-07	1.99E-08
	6	advection1D	2.37E-04	5.07E-08	4.29E-04	2.34E-09	3.18E-04	7.64E-09
		burgers1D	7.57E-03	2.08E-04	1.67E-02	1.08E-05	3.29E-01	3.41E-01
		heat1D	2.50E-05	3.29E-08	7.02E-05	3.39E-10	3.47E-05	2.92E-09
T = 2500								

Figure 4.6. Loss values for DeepONet models at 2500 training points.

			Width					
			64		128		256	
			Val Loss	Final Loss	Val Loss	Final Loss	Val Loss	Final Loss
Depth	4	advection1D	5.05E-09	3.83E-09	1.55E-09	1.27E-09	7.46E-09	7.25E-09
		burgers1D	4.72E-03	1.31E-03	1.17E-02	2.06E-03	3.29E-01	3.39E-01
		heat1D	2.12E-09	1.47E-09	8.04E-09	9.43E-10	1.97E-08	2.09E-09
	5	advection1D	2.77E-08	1.42E-08	1.09E-07	1.65E-08	8.39E-08	1.91E-08
		burgers1D	2.20E-03	4.41E-04	5.04E-03	5.09E-05	3.29E-01	3.39E-01
		heat1D	1.56E-08	6.32E-09	2.75E-07	8.60E-09	3.15E-08	3.24E-09
	6	advection1D	1.23E-06	1.02E-07	6.51E-05	3.84E-09	7.46E-07	1.70E-08
		burgers1D	1.27E-03	1.74E-04	4.15E-03	1.77E-05	3.29E-01	3.39E-01
		heat1D	1.49E-06	5.42E-08	5.63E-06	9.22E-09	8.73E-07	1.17E-08
T = 5000								

Figure 4.7. Loss values for DeepONet models at 5000 training points.

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	5.08E-06	2.22E-06	1.90E-06	9.80E-07	8.66E-07	4.24E-07
		burgers1D	3.14E-02	4.30E-05	3.31E-02	1.32E-05	2.14E-02	7.39E-06
		heat1D	1.82E-07	9.78E-08	5.26E-08	2.78E-08	2.23E-08	1.66E-08
	5	advection1D	6.29E-06	2.73E-06	1.27E-06	6.32E-07	9.63E-07	4.25E-07
		burgers1D	2.07E-02	2.21E-05	1.55E-02	7.41E-06	1.54E-02	3.70E-06
		heat1D	1.02E-07	5.85E-08	3.09E-08	1.86E-08	1.85E-08	1.34E-08
	6	advection1D	2.85E-06	1.44E-06	1.15E-06	6.38E-07	9.26E-07	5.01E-07
		burgers1D	1.61E-02	9.84E-06	1.12E-02	4.40E-06	1.19E-02	3.04E-06
		heat1D	6.07E-08	4.23E-08	4.21E-08	2.58E-08	2.70E-08	1.65E-08
Training Data = 500 F = 8								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	4.12E-06	1.94E-06	1.93E-06	1.16E-06	8.05E-07	3.68E-07
		burgers1D	3.51E-02	1.72E-05	3.28E-02	6.84E-06	2.61E-02	4.63E-06
		heat1D	8.06E-08	4.56E-08	3.11E-08	2.00E-08	1.95E-08	1.21E-08
	5	advection1D	3.28E-06	1.55E-06	1.18E-06	5.93E-07	8.22E-07	3.64E-07
		burgers1D	3.03E-02	1.19E-05	2.36E-02	3.91E-06	2.16E-02	2.96E-06
		heat1D	9.20E-08	6.33E-08	4.14E-08	2.67E-08	2.01E-08	1.70E-08
	6	advection1D	9.27E-06	2.28E-06	1.34E-06	6.46E-07	6.35E-07	3.22E-07
		burgers1D	1.89E-02	4.36E-06	1.71E-02	2.87E-06	1.66E-02	1.58E-06
		heat1D	7.46E-08	4.77E-08	5.19E-08	3.26E-08	2.31E-08	2.01E-08
T = 500 F = 12								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	4.67E-06	2.15E-06	2.31E-06	1.37E-06	2.52E-06	7.72E-07
		burgers1D	3.02E-02	1.08E-05	3.13E-02	4.77E-06	2.48E-02	2.59E-06
		heat1D	6.15E-08	4.56E-08	3.48E-08	2.35E-08	2.55E-08	1.59E-08
	5	advection1D	4.39E-06	2.14E-06	1.24E-06	6.41E-07	8.94E-07	4.19E-07
		burgers1D	2.62E-02	4.06E-06	2.34E-02	2.95E-06	2.00E-02	1.60E-06
		heat1D	8.32E-08	5.08E-08	4.60E-08	2.66E-08	2.14E-08	1.14E-08
	6	advection1D	3.48E-06	1.41E-06	8.99E-07	5.14E-07	1.04E-06	4.62E-07
		burgers1D	2.34E-02	4.46E-06	1.84E-02	1.85E-06	1.55E-02	1.19E-06
		heat1D	8.25E-08	5.77E-08	3.42E-08	1.96E-08	2.92E-08	1.75E-08
T = 500 F = 16								

Figure 4.8. Loss values for FNO models at 500 training points.

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	4.97E-07	2.32E-07	1.52E-07	9.89E-08	1.60E-07	8.49E-08
		burgers1D	7.04E-03	1.27E-05	7.76E-03	6.04E-06	6.01E-03	2.69E-06
		heat1D	1.46E-08	8.82E-09	6.89E-09	4.82E-09	5.22E-09	4.07E-09
	5	advection1D	4.31E-07	2.32E-07	1.79E-07	1.03E-07	9.37E-08	5.80E-08
		burgers1D	6.23E-03	7.08E-06	4.56E-03	3.13E-06	4.51E-03	1.89E-06
		heat1D	1.19E-08	7.41E-09	5.70E-09	3.98E-09	3.59E-09	3.28E-09
	6	advection1D	5.03E-07	2.01E-07	1.38E-07	8.04E-08	8.43E-08	4.77E-08
		burgers1D	4.42E-03	5.22E-06	2.70E-03	2.42E-06	2.60E-03	1.63E-06
		heat1D	1.66E-08	1.17E-08	7.00E-09	4.53E-09	6.56E-09	5.05E-09
T = 1000 F = 8								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	6.75E-07	2.89E-07	2.85E-07	1.40E-07	1.19E-07	7.04E-08
		burgers1D	1.29E-02	7.68E-06	1.15E-02	2.74E-06	8.43E-03	1.69E-06
		heat1D	1.44E-08	9.63E-09	6.08E-09	4.84E-09	4.71E-09	3.40E-09
	5	advection1D	5.29E-07	2.21E-07	1.57E-07	1.04E-07	1.08E-07	6.99E-08
		burgers1D	7.85E-03	3.52E-06	6.37E-03	1.63E-06	6.80E-03	1.18E-06
		heat1D	1.16E-08	8.75E-09	7.72E-09	5.78E-09	5.12E-09	4.47E-09
	6	advection1D	4.98E-07	2.07E-07	1.85E-07	1.02E-07	1.19E-07	7.26E-08
		burgers1D	5.91E-03	2.76E-06	4.24E-03	1.58E-06	3.07E-03	1.22E-06
		heat1D	1.55E-08	9.13E-09	4.96E-09	4.21E-09	3.87E-09	3.08E-09
T = 1000 F = 12								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	8.04E-07	4.11E-07	2.62E-07	1.36E-07	1.46E-07	8.83E-08
		burgers1D	1.67E-02	6.44E-06	1.18E-02	2.00E-06	1.09E-02	1.24E-06
		heat1D	1.59E-08	1.10E-08	1.07E-08	7.35E-09	3.13E-09	2.63E-09
	5	advection1D	7.72E-07	3.09E-07	1.82E-07	8.33E-08	1.05E-07	6.62E-08
		burgers1D	1.01E-02	3.26E-06	6.28E-03	1.36E-06	5.89E-03	9.11E-07
		heat1D	1.16E-08	7.41E-09	4.63E-09	3.57E-09	5.43E-09	3.80E-09
	6	advection1D	7.00E-07	2.08E-07	1.24E-07	7.44E-08	1.18E-07	7.86E-08
		burgers1D	5.78E-03	2.11E-06	3.67E-03	1.38E-06	3.93E-03	9.45E-07
		heat1D	1.88E-08	9.35E-09	8.94E-09	5.22E-09	7.08E-09	5.28E-09
T = 1000 F = 16								

Figure 4.9. Loss values for FNO models at 1000 training points.

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	5.75E-08	3.81E-08	2.66E-08	2.15E-08	2.08E-08	1.61E-08
		burgers1D	5.91E-04	4.21E-06	6.72E-04	2.28E-06	8.38E-04	1.52E-06
		heat1D	3.05E-09	2.56E-09	1.75E-09	1.31E-09	1.35E-09	1.02E-09
	5	advection1D	5.05E-08	3.35E-08	2.44E-08	1.88E-08	1.77E-08	1.33E-08
		burgers1D	3.97E-04	2.74E-06	5.45E-04	1.58E-06	3.44E-04	1.02E-06
		heat1D	3.11E-09	2.66E-09	1.80E-09	1.31E-09	1.25E-09	1.04E-09
	6	advection1D	4.69E-08	3.22E-08	2.15E-08	1.76E-08	2.16E-08	1.64E-08
		burgers1D	4.27E-04	1.90E-06	3.04E-04	1.06E-06	3.25E-04	7.43E-07
		heat1D	3.88E-09	3.44E-09	2.27E-09	1.95E-09	1.53E-09	1.57E-09
T = 2500 F = 8								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	6.71E-08	4.56E-08	3.36E-08	2.45E-08	1.72E-08	1.35E-08
		burgers1D	1.48E-03	3.66E-06	1.64E-03	2.06E-06	1.80E-03	1.39E-06
		heat1D	2.16E-09	1.85E-09	1.66E-09	1.16E-09	9.38E-10	1.03E-09
	5	advection1D	6.09E-08	4.07E-08	2.55E-08	1.95E-08	1.82E-08	1.34E-08
		burgers1D	1.46E-03	1.99E-06	9.24E-04	1.16E-06	9.48E-04	8.59E-07
		heat1D	2.53E-09	2.08E-09	1.89E-09	1.59E-09	1.41E-09	1.20E-09
	6	advection1D	5.74E-08	3.55E-08	4.28E-08	2.70E-08	2.48E-08	1.87E-08
		burgers1D	5.97E-04	1.54E-06	9.71E-04	8.74E-07	5.45E-04	7.17E-07
		heat1D	4.58E-09	3.25E-09	1.65E-09	1.74E-09	2.38E-09	1.72E-09
T = 2500 F = 12								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	6.51E-08	4.38E-08	3.75E-08	2.49E-08	1.80E-08	1.29E-08
		burgers1D	2.58E-03	3.50E-06	1.97E-03	1.46E-06	2.06E-03	1.14E-06
		heat1D	2.12E-09	1.76E-09	1.18E-09	1.14E-09	1.71E-09	1.40E-09
	5	advection1D	5.85E-08	3.90E-08	3.33E-08	2.06E-08	2.38E-08	1.84E-08
		burgers1D	1.46E-03	1.76E-06	1.25E-03	9.67E-07	1.41E-03	6.92E-07
		heat1D	3.59E-09	3.19E-09	1.47E-09	1.23E-09	2.25E-09	1.95E-09
	6	advection1D	5.50E-08	3.82E-08	3.04E-08	2.13E-08	2.40E-08	1.93E-08
		burgers1D	1.08E-03	1.24E-06	7.36E-04	9.30E-07	7.79E-04	6.82E-07
		heat1D	4.04E-09	2.92E-09	1.57E-09	1.53E-09	1.47E-09	1.43E-09
T = 2500 F = 16								

Figure 4.10. Loss values for FNO models at 2500 training points.

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	1.40E-08	1.29E-08	8.56E-09	6.68E-09	6.04E-09	5.54E-09
		burgers1D	1.13E-04	1.73E-06	1.19E-04	1.07E-06	1.18E-04	7.28E-07
		heat1D	1.08E-09	1.13E-09	1.28E-09	1.19E-09	6.91E-10	7.57E-10
	5	advection1D	1.81E-08	1.58E-08	1.51E-08	1.19E-08	6.55E-09	7.33E-09
		burgers1D	9.73E-05	1.18E-06	1.60E-04	7.96E-07	1.74E-04	5.90E-07
		heat1D	1.49E-09	1.39E-09	1.70E-09	1.32E-09	1.29E-09	1.26E-09
	6	advection1D	2.25E-08	1.86E-08	1.39E-08	1.07E-08	1.51E-08	1.03E-08
		burgers1D	1.04E-04	9.89E-07	1.82E-04	5.46E-07	1.32E-04	4.10E-07
		heat1D	1.42E-09	1.26E-09	1.80E-09	1.79E-09	1.43E-09	1.77E-09
T = 5000 F = 8								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	1.14E-08	9.66E-09	8.09E-09	7.55E-09	7.47E-09	6.18E-09
		burgers1D	3.54E-04	1.99E-06	2.76E-04	1.02E-06	3.27E-04	6.67E-07
		heat1D	1.32E-09	1.15E-09	1.07E-09	1.16E-09	1.01E-09	1.01E-09
	5	advection1D	2.19E-08	1.62E-08	1.49E-08	1.21E-08	8.53E-09	7.27E-09
		burgers1D	2.59E-04	1.15E-06	4.20E-04	7.63E-07	4.07E-04	5.58E-07
		heat1D	1.41E-09	1.40E-09	1.50E-09	1.39E-09	1.96E-09	1.77E-09
	6	advection1D	2.54E-08	1.71E-08	1.46E-08	1.10E-08	1.95E-08	1.78E-08
		burgers1D	1.57E-04	7.92E-07	3.45E-04	5.00E-07	4.11E-04	3.82E-07
		heat1D	1.17E-09	1.21E-09	1.90E-09	1.97E-09	1.76E-09	1.80E-09
T = 5000 F = 12								

			Width					
			64		96		128	
			Validation Loss	Final Loss	Validation Loss	Final Loss	Validation Loss	Final Loss
Depth	4	advection1D	1.93E-08	1.54E-08	8.49E-09	8.00E-09	8.41E-09	7.72E-09
		burgers1D	2.63E-04	1.52E-06	3.55E-04	1.02E-06	4.87E-04	7.95E-07
		heat1D	1.17E-09	1.06E-09	7.76E-10	9.49E-10	6.69E-10	7.01E-10
	5	advection1D	1.83E-08	1.51E-08	1.03E-08	9.10E-09	1.16E-08	9.05E-09
		burgers1D	2.26E-04	9.21E-07	3.65E-04	6.31E-07	3.95E-04	4.67E-07
		heat1D	1.23E-09	1.28E-09	1.10E-09	1.21E-09	1.37E-09	1.40E-09
	6	advection1D	2.24E-08	1.61E-08	1.63E-08	1.27E-08	1.44E-08	1.08E-08
		burgers1D	2.51E-04	7.75E-07	1.94E-04	4.88E-07	3.12E-04	3.52E-07
		heat1D	1.89E-09	1.74E-09	1.68E-09	1.59E-09	1.72E-09	1.69E-09
T = 5000 F = 16								

Figure 4.11. Loss values for FNO models at 5000 training points.

4.4.2 Solution Accuracy

Solution accuracy can be observed through similar plots to those used in Chapter 3. For this comparison, we have selected three solution plots for both neural operator networks for

each of the three PDEs that had a similar amount of trainable parameters and training data. In this case, the DeepONet models chosen for the three PDEs had a depth of 6 and a width of 256 which resulted in 920,832 trainable parameters. For comparison, the FNO models chosen had depths of 6, widths of 128, and cutoff values of 8 which resulted in 919,169 trainable parameters. The side-by-side comparisons can be seen in Figures 4.12-4.14. As could be inferred from the training and validation loss tables, both neural operator networks performed well for the 1D advection and heat equations, but the FNO models performed significantly better for the 1D Burgers equation.

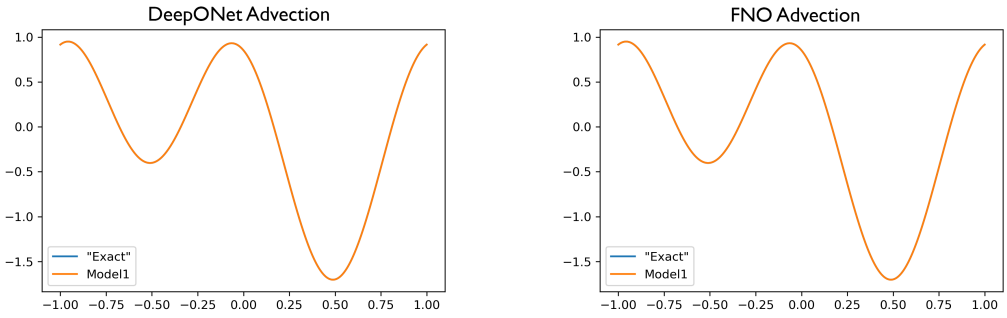

Figure 4.12. 1D advection solution plots for a DeepONet model (left) and an FNO model at similar training parameters (right).

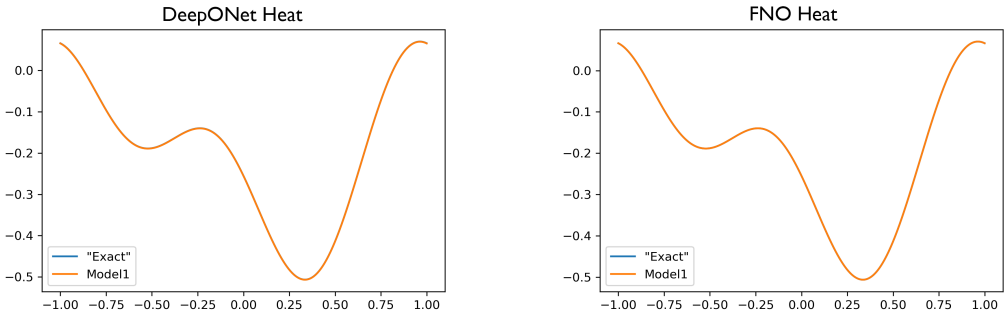

Figure 4.13. 1D heat solution plots for a DeepONet model (left) and an FNO model at similar training parameters (right).

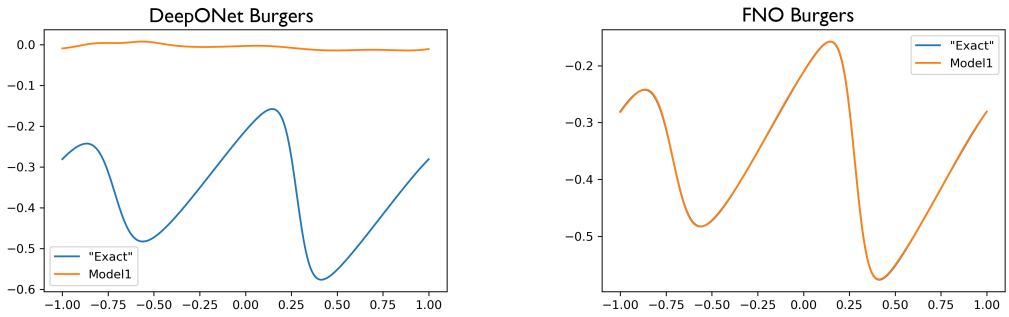

Figure 4.14. 1D Burgers solution plots for a DeepONet model (left) and an FNO model at similar training parameters (right).

4.5 Conclusions

The biggest takeaway from this set of comparison experiments is that both neural operator architectures achieved adequate loss values given enough training data when dealing with simple linear PDEs. Although neither performed as well for the nonlinear Burgers equation, FNOs were more successful at handling this problem.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion

The primary objectives of this thesis were to assess the potential advantages of residual layers over fully-connected layers in physics-informed machine learning and to compare the efficiency of the popular DeepONet and Fourier neural operator (FNO) approaches to operator learning on both linear and nonlinear problems.

In the first set of experiments, we compared a handful of fully-connected networks to networks that contained residual layers with an identical amount of trainable parameters to determine if adding residual layers added any benefits in training time or accuracy in our PINN models. Based on our observations, we determined that while adding residual layers may offer marginal benefits in terms of training time, it does not produce models that are substantially more or less accurate than those based on fully-connected layers.

In the second set of experiments, we compared two neural operators, DeepONets and FNOs, to see which, if either, architecture performs better at learning the solution operators for some simple PDEs. We concluded that given enough training data, both neural operators could achieve optimal training loss values with great accuracy for the linear heat and advection equations. Still, FNOs performed better for the 1D Burgers equation.

The next step is to take what we have learned from our experiments and create machine-learning models that can be inserted into existing physical models to see if similar or improved results can be achieved. This would involve training networks using PDEs with a much greater complexity which remains unknown whether or not these architectures can learn given the simple PDEs observed in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] M. A. Milley, "Milley: Artificial intelligence could change warfare," https://www.ausa.org/news/milley-artificial-intelligence-could-change-warfare, 2018.

[2] A. Corrin, "Dod spent $7.4 billion on big data, AI and the cloud last year. is that enough?" https://www.c4isrnet.com/it-networks/2017/12/06/dods-leaning-in-on-artificial-intelligence-will-it-be-enough/, 2017.

[3] M. Lee, R. Valisetty, A. Breuer, K. Kirk, B. Panneton, and S. Brown, "Current and future applications of machine learning for the us army," *U.S. Army Research Laboratory Journal*, April 2018.

[4] D. Randall, M. Khairoutdinov, A. Arakawa, and W. Grabowski, "Breaking the cloud parameterization deadlock," *Bull. Amer. Meteorol. Soc.*, vol. 84, pp. 1547–1564, 2003 [Online]. Available: https://doi.org/10.1175/bams-84-11-1547

[5] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis, "Could machine learning break the convection parameterization deadlock?" *Geophys. Res. Lett.*, vol. 45, pp. 5742–5751, 2018 [Online]. Available: https://doi.org/10.1029/2018gl078202

[6] A. Gettelman *et al.*, "Machine learning the warm rain process," *J. Adv. Model. Earth. Sys.*, vol. 13, pp. e2020MS002 268:1–23, 2021 [Online]. Available: https://doi.org/10.1029/2020MS002268

[7] Y. Han, G. J. Zhang, X. Huang, and Y. Wang, "A moist physics parameterization based on deep learning," *J. Adv. Model. Earth. Sys.*, vol. 12, pp. e2020MS002 076:1–20, 2020 [Online]. Available: https://doi.org/10.1029/2020ms002076

[8] S. Rasp, M. S. Pritchard, and P. Gentine, "Deep learning to represent subgrid processes in climate models," *Proc. Natl. Acad. Sci. USA*, vol. 115, pp. 9684–9689, 2018 [Online]. Available: https://doi.org/10.1073/pnas.1810286115

[9] K. D. Foote, "A brief history of machine learning," https://www.dataversity.net/a-brief-history-of-machine-learning/, 2021.

[10] C. Higham and D. Higham, "Deep learning: An introduction for applied mathematicians," *SIAM Rev.*, vol. 61, pp. 860–891, 2019 [Online]. Available: https://doi.org/10.1137/18M1165748

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control. Signal. Sys.*, vol. 2, pp. 303–314, 1989 [Online]. Available: https://doi.org/10.1007/BF02551274

[12] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991 [Online]. Available: https://doi.org/10.1016/0893-6080(91)90009-T

[13] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review," *Int. J. Automat. Comput.*, vol. 14(5), pp. 503–519, 2017 [Online]. Available: https://doi.org/10.1007/s11633-017-1054-2

[14] Q. Gong, W. Kang, and F. Fahroo, "Approximation of compositional functions with ReLU neural networks," *Syst. Control Lett.*, vol. 175, p. 105508, 2023 [Online]. Available: https://doi.org/10.1016/j.sysconle.2023.105508

[15] W. Kang and Q. Gong, "Feedforward neural networks and compositional functions with applications to dynamical systems," *SIAM J. Control Opt.*, vol. 60, no. 2, pp. 786–813, 2022 [Online]. Available: https://doi.org/10.1137/21M1391596

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. http://www.deeplearningbook.org.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR 2015), Conference Track Proceedings*, 2015 [Online]. Available: https://arxiv.org/abs/1412.6980

[18] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *J. Mach. Learn. Res.*, pp. 1–43, 2018.

[19] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015 [Online]. Available: https://www.tensorflow.org/

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019 [Online]. Available: https://doi.org/10.1016/j.jcp.2018.10.045

[21] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and what's next," *J. Sci. Comput.*, vol. 92, pp. 1–62, 2022 [Online]. Available: https://doi.org/10.1007/s10915-022-01939-z

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[23] B. Tang, "Orthogonal array-based Latin hypercubes," *J. Amer. Stat. Assoc.*, vol. 88, pp. 1392–1397, 1993 [Online]. Available: https://doi.org/10.1080/01621459.1993.10476423

[24] I. Molybog *et al.*, "A theory on Adam instability in large-scale machine learning," *arXiv*, 2023. `arXiv:2304.09871 [cs.LG]`.

[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, vol. 9, pp. 249–256.

[26] L. Lu *et al.*, "A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data," *Comput. Methods Appl. Mech. Eng.*, vol. 393, p. 114778, 2022 [Online]. Available: https://doi.org/10.1016/j.cma.2022.114778

[27] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nat. Mach. Intell.*, vol. 3, pp. 218–229, 2021 [Online]. Available: https://doi.org/10.1038/s42256-021-00302-5

[28] T. A. Driscoll, N. Hale, and L. N. Trefethen, Eds., *Chebfun Guide*. Oxford: Pafnuty Publications, 2014.

[29] S. Filip, A. Javeed, and L. N. Trefethen, "Smooth random functions, random odes, and Gaussian processes," *SIAM Rev.*, vol. 61, pp. 185–205, 2019 [Online]. Available: https://doi.org/10.1137/17m1161853

[30] H. Montanelli and N. Bootland, "Solving periodic semilinear stiff pdes in 1d, 2d and 3d with exponential integrators," *Math. Comput. Simulation*, vol. 178, pp. 307–327, 2020 [Online]. Available: https://doi.org/10.1016/j.matcom.2020.06.008

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California