

Solutions to Elliptic and Parabolic Problems via Finite Difference Based Unsupervised Small Linear Convolutional Neural Networks

Adrian Celaya^{a,b}, Keegan Kirk^a, David Fuentes^b, Beatrice Riviere^a

^a*Rice University, Houston, 77005, TX, USA*

^b*The University of Texas MD Anderson Cancer Center, Houston, 77003, TX, USA*

Abstract

In recent years, there has been a growing interest in leveraging deep learning and neural networks to address scientific problems, particularly in solving partial differential equations (PDEs). However, many neural network-based methods like PINNs rely on auto differentiation and sampling collocation points, leading to a lack of interpretability and lower accuracy than traditional numerical methods. As a result, we propose a fully unsupervised approach, requiring no training data, to estimate finite difference solutions for PDEs directly via small linear convolutional neural networks. Our proposed approach uses substantially fewer parameters than similar finite difference-based approaches while also demonstrating comparable accuracy to the true solution for several selected elliptic and parabolic problems compared to the finite difference method.

Keywords: Convolutional neural networks, unsupervised learning, partial differential equations, finite difference

1. Introduction

Partial differential equations (PDEs) of elliptic and parabolic type are ubiquitous in the mathematical modeling of many physical phenomena and thus see wide application to many real-world problems. Classical numerical methods (i.e., finite difference and element methods) introduce a computational mesh over which we define representations of differential operators with matrices. This results in large linear systems whose efficient solution presents several computational challenges. Physics informed neural networks

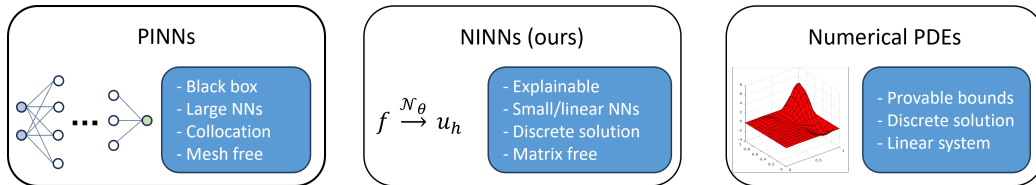


Figure 1: Properties of classical PINNs, numerics informed neural networks (NINNs), and numerical PDEs.

(PINNs) have recently gained popularity in solving PDEs [1, 2, 3, 4, 5, 6]. This approach uses the universal approximation property of deep neural networks to train a surrogate model (i.e., the neural network) of the solution to a given PDE.

A vital aspect of PINNs is using auto differentiation to compute a residual-based loss function for a set of sampled collocation points [7]. While PINNs represent a mesh-free solution and have shown promise in multiple fields like biology [8, 9], meteorology [10], and optimal control [11, 12], this reliance on auto differentiation and sampling results in a lack of interpretability and lower accuracy than traditional numerical methods [13]. Choosing collocation points involves sampling random points from a uniform distribution on the computational domain, and it is unclear how many points are required to achieve acceptable results for a given PDE [13, 7, 14]. Additionally, while networks can express very complex functions, determining the appropriate architecture and number of parameters to solve a specific PDE can be challenging [15, 16, 17]. If a specific PDE requires a neural network with many parameters, it will increase the memory and time costs required for training. Finally, the generalization capability of PINNs to points outside their training domains is unclear, and a topic of ongoing research [18].

While traditional numerical methods (such as finite difference and finite element methods) are computationally expensive, they are highly structured and can guide the design of neural network-based methods so that they output explainable solutions and are more computationally efficient. If we constrain a neural network-based approach to mirror a numerical method, it is reasonable to expect solutions from the neural network to be close to those of a numerical method with known error bounds. Additionally, the structure of numerical methods makes it possible to get away with using simpler neural networks that converge to small loss values. Examples of such neural network-based methods include the Deep Ritz and Deep Galerkin

methods [19, 20, 21]. Both of these methods incorporate aspects of finite element methods, allowing for faster convergence during training by utilizing the structure and properties of the numerical method that inspired them.

This work proposes a finite difference-based approach to approximating solutions to elliptic and parabolic PDEs that utilizes small, linear convolutional neural networks (CNNs). Figure 1 displays the properties of our proposed method which we call a “numerics informed neural network” (NINN), classical PINNs, and numerical PDEs. Our choice of architecture, inspired by the PocketNet paradigm [22], uses CNNs that closely mirror geometric multigrid methods to almost exactly recover finite difference solutions with fewer parameters than other CNN-based methods. To the best of our knowledge, we are also the first to propose using linear CNNs, which do not use non-linear activation functions, to learn finite difference solutions.

Finite difference methods have inspired the construction of various neural networks [13, 23], but these methods require training data. The recent work [24] solves elliptic PDEs with constant diffusion, and it shares the following similarities with our proposed numerical method: no training data is needed, and the loss function is derived from the five-point stencil of the finite difference approximation. However, our loss function is different as it incorporates the Dirichlet boundary condition in a weighted fashion, whereas [24] directly enforces the boundary condition outside of the loss function (more details in Section 4). We also define the method for elliptic PDEs with non-constant diffusion coefficients and extend it to time-dependent problems. Another key feature of our method is using small neural networks, which makes it computationally efficient.

An outline of the paper is as follows. Section 2 introduces the model problem, the network architecture and the loss function. Section 3 describes the algorithms for both steady-state and time-dependent problems. Results and discussion are presented in Section 4. Conclusions follow.

2. Material and Methods

We propose a fully unsupervised method for estimating finite differences solutions to partial differential equations via convolutional neural networks. In contrast to existing deep learning-based methods, our approach is fully unsupervised. In other words, our approach does not require training data and estimates the solution to a given PDE directly via the optimization process, also called the training process.

2.1. Elliptic Problems

Let $\Omega \subset \mathbb{R}^2$ be an open set. Given a function $f : \Omega \rightarrow \mathbb{R}$ and diffusion coefficients κ , we consider the 2D Poisson problem with solution $u : \bar{\Omega} \rightarrow \mathbb{R}$ such that

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \text{ in } \Omega, \\ u &= g \text{ on } \partial\Omega. \end{aligned} \tag{1}$$

We begin with the case when κ is constant and assume that $\kappa = 1$. For readability, we assume that Ω is the square domain $(0, L)^2$, that is partitioned into a uniform grid of $N \times N$ squares with size $h = L/N$. The methodology presented in the paper can be easily extended to rectangular domains. We first recall the standard finite difference method based on the five-point stencil, applied to (1). Let $\mathcal{V}^0$ (resp. $\mathcal{V}^\partial$) be the set of indices (i, j) such that the point $(x_i, y_j) = (ih, jh)$ belongs to the interior (resp. boundary) of Ω .

$$\frac{4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}}{h^2} = f(x_i, y_j), \quad \forall (i, j) \in \mathcal{V}^0, \tag{2}$$

$$u_{i,j} = g(x_i, y_j), \quad \forall (i, j) \in \mathcal{V}^\partial. \tag{3}$$

The finite difference solution u_h is a vector with entries $u_{i,j}$ satisfying (2)-(3) and it is known that for smooth enough solutions u and small enough h , the value $u_{i,j}$ is a good approximation of $u(x_i, y_j)$. One can rewrite (2) in terms of a discrete convolutional operator $\star$ and convolutional kernel K_Δ defined as:

$$K_\Delta = \frac{1}{h^2} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{4}$$

We use the shorthand notation $f_{i,j} = f(x_i, y_j)$. The equivalent form for (2) is

$$(K_\Delta \star u_h)_{i,j} = f_{i,j}, \quad \forall (i, j) \in \mathcal{V}^0. \tag{5}$$

For a given kernel $K \in \mathbb{R}^{3 \times 3}$, the convolution operator $\star$ is defined by

$$(K \star u_h)_{i,j} = \sum_{p=-1}^1 \sum_{q=-1}^1 K_{p,q} u_{i+p,j+q}. \tag{6}$$

With this convolution-based discretization, we can now reformulate the finite difference approximation of (1) as a convex optimization problem given by

$$\arg \min_{u_h} \sum_{(i,j) \in \mathcal{V}^0} \left((K_\Delta \star u_h)_{i,j} - f_{i,j} \right)^2 + \sum_{(i,j) \in \mathcal{V}^\partial} \left((u_h)_{i,j} - g_{i,j} \right)^2. \quad (7)$$

We obtain an approximate solution $\hat{u}$ to (7) by training a neural network $\mathcal{N}_\theta : \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times N}$ with trainable parameters $\theta \in \mathbb{R}^M$. We propose to train $\mathcal{N}_\theta$ using the unsupervised loss function $\mathcal{L}_\alpha : \mathbb{R}^{N \times N} \rightarrow \mathbb{R}$ given by

$$\mathcal{L}_\alpha(\hat{u}) = \alpha \sum_{(i,j) \in \mathcal{V}^0} \left((K_\Delta \star \hat{u})_{i,j} - f_{i,j} \right)^2 + (1 - \alpha) \sum_{(i,j) \in \mathcal{V}^\partial} (\hat{u}_{i,j} - g_{i,j})^2, \quad (8)$$

where $\alpha = h^2/4$ is a weighting term. The idea behind this loss function is to directly estimate the finite difference approximation to (1) without using any training data. Specifically, we use f as the input to the CNN and we perform $\mathcal{M}$ updates of the parameters in the neural network $\mathcal{N}_\theta$, using the loss function defined above. Algorithm 1 in Section 3 outlines this training procedure in more detail.

In the case where κ is piecewise constant, $\kappa = (\kappa_{i,j})$, we use the following finite difference discretization

$$\begin{aligned} & -\frac{1}{h^2} \left(\kappa_{i+\frac{1}{2},j} (u_{i+1,j} - u_{i,j}) - \kappa_{i-\frac{1}{2},j} (u_{i,j} - u_{i-1,j}) \right) \\ & -\frac{1}{h^2} \left(\kappa_{i,j+\frac{1}{2}} (u_{i,j+1} - u_{i,j}) - \kappa_{i,j-\frac{1}{2}} (u_{i,j} - u_{i,j-1}) \right) = f_{i,j}, \end{aligned} \quad (9)$$

where the interface values $\kappa_{i+\frac{1}{2},j}$, $\kappa_{i-\frac{1}{2},j}$, $\kappa_{i,j+\frac{1}{2}}$, and $\kappa_{i,j-\frac{1}{2}}$ are the harmonic averages of $\kappa_{i,j}$ given by

$$\begin{aligned} \kappa_{i+\frac{1}{2},j} &= \frac{2}{\frac{1}{\kappa_{i+1,j}} + \frac{1}{\kappa_{i,j}}} & \kappa_{i-\frac{1}{2},j} &= \frac{2}{\frac{1}{\kappa_{i,j}} + \frac{1}{\kappa_{i-1,j}}} \\ \kappa_{i,j+\frac{1}{2}} &= \frac{2}{\frac{1}{\kappa_{i,j+1}} + \frac{1}{\kappa_{i,j}}} & \kappa_{i,j-\frac{1}{2}} &= \frac{2}{\frac{1}{\kappa_{i,j}} + \frac{1}{\kappa_{i,j-1}}}. \end{aligned} \quad (10)$$

The interface values necessitate the imposition of a half-grid (also called dual grid) onto our original grid. As a result, we modify our neural network prediction via a dilation operator $D : \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{(2N-1) \times (2N-1)}$ such that

$$D(\hat{u})_{i,j} = \begin{cases} \hat{u}_{1+\frac{i-1}{2}, 1+\frac{j-1}{2}}, & \text{if } i \text{ and } j \text{ are odd} \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

Additionally, we map κ from the original grid to the dual grid by applying the following steps:

Step 1: apply the dilation operator to obtain $\kappa^* \in \mathbb{R}^{(2N-1) \times (2N-1)}$:

$$\kappa^* = D(\kappa). \quad (12)$$

Step 2: update the values of κ^* on the half-grid:

$$\kappa_{i,j}^* = \begin{cases} 2 \left(\frac{1}{\kappa_{i-1,j}^*} + \frac{1}{\kappa_{i+1,j}^*} \right)^{-1}, & \text{if } i \text{ odd and } j \text{ even} \\ 2 \left(\frac{1}{\kappa_{i,j-1}^*} + \frac{1}{\kappa_{i,j+1}^*} \right)^{-1}, & \text{if } i \text{ even and } j \text{ odd} \end{cases} \quad (13)$$

Step 3: set to zero the values of κ^* that would correspond to the values on the original grid:

$$\tilde{\kappa} = (\mathbf{1}_{(2N-1) \times (2N-1)} - D(\mathbf{1}_{N \times N})) \odot \kappa^*. \quad (14)$$

In the equation above, $\mathbf{1}_{N \times N}$ is an $N \times N$ matrix with all entries equal to one, and $\odot$ is the Hadamard or pointwise product.

We now introduce the following convolution kernels

$$\begin{aligned} T_{\uparrow} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} & P_{\uparrow} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ T_{\leftarrow} &= -T_{\uparrow}^T & P_{\leftarrow} &= P_{\uparrow}^T \\ T_{\downarrow} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} & P_{\downarrow} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ T_{\rightarrow} &= -T_{\downarrow}^T & P_{\rightarrow} &= P_{\downarrow}^T, \end{aligned} \quad (15)$$

which will allow us to rewrite (9) as follows:

$$\begin{aligned} & -\frac{1}{h^2} ((T_{\uparrow} \star D(u)) \odot (P_{\uparrow} \star \tilde{\kappa}) - (T_{\downarrow} \star D(u)) \odot (P_{\downarrow} \star \tilde{\kappa}) \\ & + (T_{\rightarrow} \star D(u)) \odot (P_{\rightarrow} \star \tilde{\kappa}) - (T_{\leftarrow} \star D(u)) \odot (P_{\leftarrow} \star \tilde{\kappa}))_{i,j} = f_{i,j}, \forall i, j \in \mathcal{V}^0, \end{aligned} \quad (16)$$

which is then used in the first term of the loss function (8) to train our neural network. The same training process outlined in Algorithm 1 applies here. Note that in (16), the convolution operator $\star$ uses a stride of two because of the dual grid and the use of larger convolution kernels.

2.2. Parabolic Problems

We generalize the method described in the previous section to time-dependent problems. We propose a modified loss function that incorporates the backward Euler method to address the time derivative. Consider the following parabolic problem on a square domain Ω and for the time interval $[t_0, T]$.

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= f \quad \text{in } \Omega \times (t_0, T], \\ u &= g \quad \text{on } \partial\Omega \times (t_0, T], \\ u(\cdot, t_0) &= u^0 \quad \text{in } \Omega \times \{t_0\}. \end{aligned} \tag{17}$$

Let $\tau > 0$ be the time step size. At each time step, $t^n = n\tau$, the finite difference solution denoted by u_h^n solves the following optimization problem

$$\begin{aligned} \arg \min_{u_h^n} \sum_{(i,j) \in \mathcal{V}^0} \left(u_{i,j}^n - u_{i,j}^{n-1} - \tau \left(f_{i,j}^n - (K_\Delta \star u_h^n)_{i,j} \right) \right)^2 \\ + \sum_{(i,j) \in \mathcal{V}^\partial} (u_{i,j}^n - g_{i,j}^n)^2. \end{aligned} \tag{18}$$

Above we use the short-hand notation $f_{i,j}^n = f(x_i, y_j, t^n)$ and $g_{i,j}^n = g(x_i, y_j, t^n)$. Here, the unknowns $u_{i,j}^n$ are the approximations of the solution u evaluated at the point (x_i, y_j) and time t^n . We then use the following loss function to train our CNN at each time step

$$\begin{aligned} \mathcal{L}_{\alpha, \tau}(\hat{u}^n, \hat{u}^{n-1}, t^n) &= \alpha \sum_{(i,j) \in \mathcal{V}^0} \left(\hat{u}_{i,j}^n - \hat{u}_{i,j}^{n-1} - \tau \left(f_{i,j}^n - (K_\Delta \star \hat{u}^n)_{i,j} \right) \right)^2 \\ &+ (1 - \alpha) \sum_{(i,j) \in \mathcal{V}^\partial} (\hat{u}_{i,j}^n - g_{i,j}^n)^2. \end{aligned} \tag{19}$$

Again, we will choose the weight $\alpha = h^2/4$. The key difference between (8) and (19) is that we minimize at each time step. The input to the CNN at each time step is the previous solution $\hat{u}^{n-1}$ and the output is the solution at the time step t^n . Algorithm 2 in Section 3 outlines this training procedure in more detail.

2.3. Network Architecture

We select as our architecture the widely used U-Net [25]. The U-Net was initially designed for image segmentation tasks but is successful in several scientific machine learning tasks [26, 27, 28]. The architecture consists of convolution block operators (blocks), a downsampling (or encoding) path, and an upsampling (or decoding) path. A channel-wise concatenation operation links each layer in the downsampling and upsampling paths. Each block consists of two 2D convolution layers. Unlike most implementations of the U-Net architecture [25, 29, 30], we omit normalization (i.e., batch or instance normalization) and do not apply a non-linear activation function like ReLU and use the identity function instead (see Section 4). Additionally, we use the PocketNet approach proposed in [22] and leave the number of feature maps (channels) at each resolution within our architecture constant, namely 32. Figure 2 sketches the U-Net architecture for different network depths. We define the depth, d , of each network to be the number of downsampling operations present in the network. The size of the feature tensor is halved from depth d to depth $d + 1$.

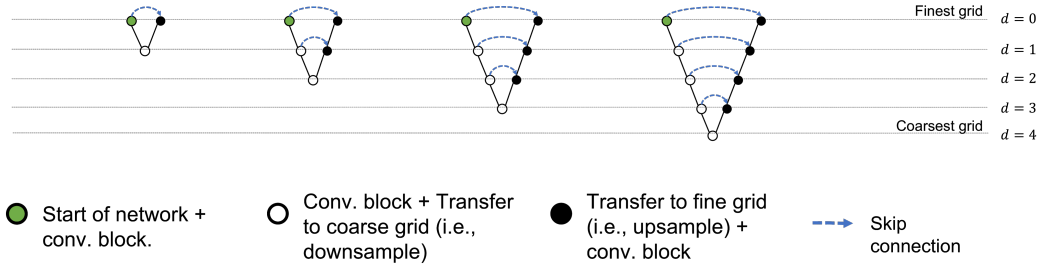


Figure 2: Sketch of U-Net architecture for different network depths: $0 \leq d \leq 4$.

2.4. Data

2.4.1. Elliptic Problem Data

We test our proposed approach for elliptic problems on four different test cases. Unless otherwise specified, the domain is the unit square.

The Bubble Function. In the first test case, (1) is solved with data chosen such that the exact solution is the bubble function

$$u(x, y) = x(x - 1)y(y - 1). \quad (20)$$

This function is symmetric, and we apply homogeneous zero Dirichlet boundary conditions. Note that for this function, one can show that the five-point stencil scheme given by (2) applied to the bubble function gives exactly $-\Delta u$ evaluated at the grid point (x_i, y_j) , which implies that the finite difference approximation is exact.

The “Peak” Function. In the second case, we want to test the effectiveness of our strategy when the solution has high gradients and is non-symmetric. Hence, we select data such that the exact solution is the “peak” function

$$u(x, y) = 0.0005 (x(x - 1)y(y - 1))^2 e^{10x^2 + 10y}. \quad (21)$$

The Exponential-Trigonometric Function. We test our approach on a function with non-homogeneous Dirichlet boundary conditions in the third case. We select data such that the exact solution is the exponential-trigonometric function given by

$$u(x, y) = e^{-x^2 - y^2} \sin(3\pi x) \sin(3\pi y) + x. \quad (22)$$

Non-Constant and Discontinuous Diffusion. To test how our approach fairs on problems with low regularity, we consider an example with a non-constant, discontinuous diffusion coefficient taken from [31]. The problem is posed on the square domain $\Omega = (-1, 1)^2$ which we divide into four subdomains Ω_i corresponding to the quadrants of the Cartesian plane. On each subdomain, κ is constant and takes the values $\kappa_1 = \kappa_3 = 5$ and $\kappa_2 = \kappa_4 = 1$. The exact solution takes the form

$$r^\beta (a_i \sin(\beta\theta) + b_i \cos(\beta\theta)), \quad (23)$$

where (r, θ) are the polar coordinates of a given point in Ω , a_i, b_i are constants that depend on the subdomains (see [31] for exact values). This discontinuity in the diffusion coefficient introduces a singularity in the solution at the origin, namely the function belongs to the Sobolev space $H^{1+\beta}(\Omega)$, with $\beta \approx 0.5354$.

2.4.2. Parabolic Problem Data

We test our proposed approach for parabolic problems on three different test cases.

The Trigonometric Function. In the first two cases, (17) is solved with data such that the exact solution is given by the trigonometric function

$$u(x, y, t) = \cos(t) \sin(n\pi x) \sin(n\pi y). \quad (24)$$

In the first case, we select $n = 1$, which gives a symmetric function with a single peak. In the second case, we set $n = 4$, which results in 16 peaks and troughs in our domain.

The Gaussian Function. In the third case, we test our approach on a Gaussian function centered on the point $(\frac{1}{2}, \frac{1}{2})$

$$u(x, y, t) = \cos(t) e^{-50((2x-1)^2 + (2y-1)^2)}. \quad (25)$$

2.5. Training and Testing Protocols

We use the Adam optimizer [32] and 5×5 convolutional kernels in each layer. The initial learning rate is 0.001 for the steady-state elliptic problems and 0.0001 for parabolic problems. During training, we use L2 regularization with a penalty of 10^{-7} and the norm of the network's gradient is clipped so that it is no greater than 10^{-2} . Our models are implemented in Python using TensorFlow (v2.12.0) and trained on an NVIDIA Quadro RTX 6000 GPU [33]. All network weights are initialized using the default TensorFlow initializers. All other hyperparameters are left at their default values. The code for our network architecture is available at <https://github.com/aecelaya/pde-nets>.

To assess the accuracy of our predicted solutions, we use the following norms of the error between the exact solution u and its approximate $\hat{u}$ for steady-state problems:

$$\|u - \hat{u}\|_{2,h} = h \sqrt{\sum_{(i,j) \in \mathcal{V}^0 \cup \mathcal{V}^\partial} (u(x_i, y_j) - \hat{u}_{i,j})^2} \quad (26)$$

$$\|u - \hat{u}\|_\infty = \max_{(i,j) \in \mathcal{V}^0 \cup \mathcal{V}^\partial} |u(x_i, y_j) - \hat{u}_{i,j}|. \quad (27)$$

For time-dependent problems, similar errors are computed at the final time T .

3. Calculation

We use Algorithm 1 to approximate a solution $\hat{u}$ to (1). In this algorithm, we start by initializing the neural network $\mathcal{N}_\theta$ with trainable parameters θ . In this case, we use the U-Net architecture described in Section 2.3. We set the maximum number of optimization steps $\mathcal{M}$ as the stopping criterion for the algorithm. We then begin the minimization (or training) process by generating a prediction from our neural network with the source term f as the input. Given the current prediction, we compute the loss value and update the network weights via backpropagation. If the loss value from the current prediction is less than the previous best value, we update our best loss and save the current prediction. Algorithm 1 fully outlines this procedure.

Algorithm 1 Unsupervised CNN Training For (1)

Input Right-hand side f and boundary condition g
Output Approximate solution to (7), $\hat{u}_*$

- 1: Randomly initialize the network parameters θ
- 2: Set maximum iterations $\mathcal{M}$
- 3: $k \leftarrow 0$ ▷ Iteration counter
- 4: $\ell_* \leftarrow +\infty$ ▷ Store smallest loss value
- 5: **while** $k < \mathcal{M}$ **do**
- 6: $\hat{u} \leftarrow \mathcal{N}_\theta(f)$ ▷ Get network prediction
- 7: $\ell \leftarrow \mathcal{L}_\alpha(\hat{u})$ ▷ Compute loss
- 8: Update θ using backpropagation on ℓ
- 9: **if** $\ell < \ell_*$ **then**
- 10: $\hat{u}_* \leftarrow \hat{u}$ ▷ Save best prediction
- 11: $\ell_* \leftarrow \ell$ ▷ Update smallest loss value
- 12: $k \leftarrow k + 1$

We use Algorithm 2 to approximate a solution $\hat{u}^{N_T}$ to (17), where N_T is an integer such that $T = N_T\tau$. Like with Algorithm 1, we randomly initialize a U-Net and set a maximum number of optimization steps as a stopping criterion. However, in this case, the maximum number of steps applies to each time step. We then set the initial condition as the previous solution. We apply nearly the same steps for each time step as with Algorithm 1. Namely, we produce a candidate prediction from our neural network with the previous solution as the input. We compute the loss value and update the network weights via backpropagation on the loss. If the loss value is lower

than the previously observed lowest loss value, then we update our lowest loss value and save the current prediction. At the end of this process, we set the previous solution to the best current solution. This process continues for every time step. Algorithm 2 fully outlines this procedure. Note that in this algorithm, we do not reinitialize our neural network weights at each time step. Instead, we use the previous network configuration as the initial state for the next time step.

Algorithm 2 Unsupervised CNN Training For (17)

Input Number of time steps N_T , final time T , initial condition u^0 , right hand side f , and boundary condition g
Output Approximate solution to (18) at final time $\hat{u}^{N_T}$

- 1: Randomly initialize the network parameters θ
- 2: Set maximum iterations $\mathcal{M}$
- 3: $\tau \leftarrow T/N_T$ ▷ Initialize time step size
- 4: $n \leftarrow 1$ ▷ Time step counter
- 5: **while** $n \leq N_T$ **do**
- 6: $k \leftarrow 0$ ▷ Iteration counter
- 7: $\ell_* \leftarrow +\infty$ ▷ Store smallest loss value
- 8: **while** $k < \mathcal{M}$ **do**
- 9: $w \leftarrow \mathcal{N}_\theta(\hat{u}^{n-1})$ ▷ Get network prediction
- 10: $\ell \leftarrow \mathcal{L}_{\alpha, \tau}(w, \hat{u}^{n-1}, n\tau)$ ▷ Compute loss
- 11: Update θ using backpropagation on ℓ
- 12: **if** $\ell < \ell_*$ **then**
- 13: $\hat{u}^n \leftarrow w$ ▷ Save best prediction
- 14: $\ell_* \leftarrow \ell$ ▷ Update smallest loss value
- 15: $k \leftarrow k + 1$
- 16: $n \leftarrow n + 1$

4. Results and Discussion

Using the methods described in Section 2 and Algorithm 1, we approximate the finite difference solution to the elliptic problems defined in Section 2.4.1 with constant diffusion coefficients (i.e., $\kappa = 1$). Table 1 shows the accuracy of our unsupervised predictions for a varying grid size and number of optimization steps. We fix the depth of our U-Net architecture to three in

this case. For reference, Table 3 shows the accuracy of the finite difference method for varying grid sizes. For the peak and exponential-trigonometric functions, our approach almost exactly recovers the finite difference solution. We do not, however, see this for the bubble function example. In that case, the solution of the finite difference method is exact. Under our selected settings (i.e., depth and optimization steps), our method stops at an error of approximately 10^{-6} in the $\|\cdot\|_{2,h}$ norm for the bubble function example. The cause of this discrepancy between our method applied to the bubble function and the finite difference solution is due mostly to the fact that single-precision is used to train the neural networks. Values of the order 10^{-6} mean that we are very close to machine epsilon (10^{-7}). Another factor is the use of stochastic, gradient-based optimizers. While the Adam optimizer is not stochastic in this case since we are not training with mini-batches of data, it is still true that we are using non-optimal step sizes. Therefore, we hypothesize that the method is getting stuck at errors $\mathcal{O}(10^{-6})$ instead of reaching machine epsilon values $\mathcal{O}(10^{-7})$ in single-precision.

Figure 3 shows the true solution, predicted solution, and absolute difference for the bubble, peak, and exponential-trigonometric cases on a 128×128 grid. This figure shows that for every case, our unsupervised algorithm produces visually indistinguishable solutions from the true solution in each case. To give more insight into our method’s performance, we show the loss function for one of the test cases (the bubble function) in Figure 4. Here, we observe that our loss function reaches values that are machine epsilon for single precision (on the order of 10^{-7}). Similar loss function values are observed for the other cases, indicating that we have reached convergence.

We also want to study the network depth’s effect on our predictions’ accuracy. Table 2 shows the accuracy of our unsupervised neural network predictions for varying grid sizes and U-Net depths. We set the number of optimization steps to 4,000. Here, we see that the depth of the U-Net architectures does not appear to have a significant effect on the accuracy of our predictions. The only notable exceptions are for the bubble and exponential-trigonometric functions on a 128×128 grid with a depth equal to two. This discrepancy may be caused by the network not capturing sufficiently rich features on the finer grid. The fact that we see a decrease in the errors between our predictions and the true solution to those comparable to the finite difference method as we increase the depth to three or greater supports this hypothesis. Note that for coarser grids (i.e., fewer grid points), we do not test higher network depths since the size of the features (the output of each

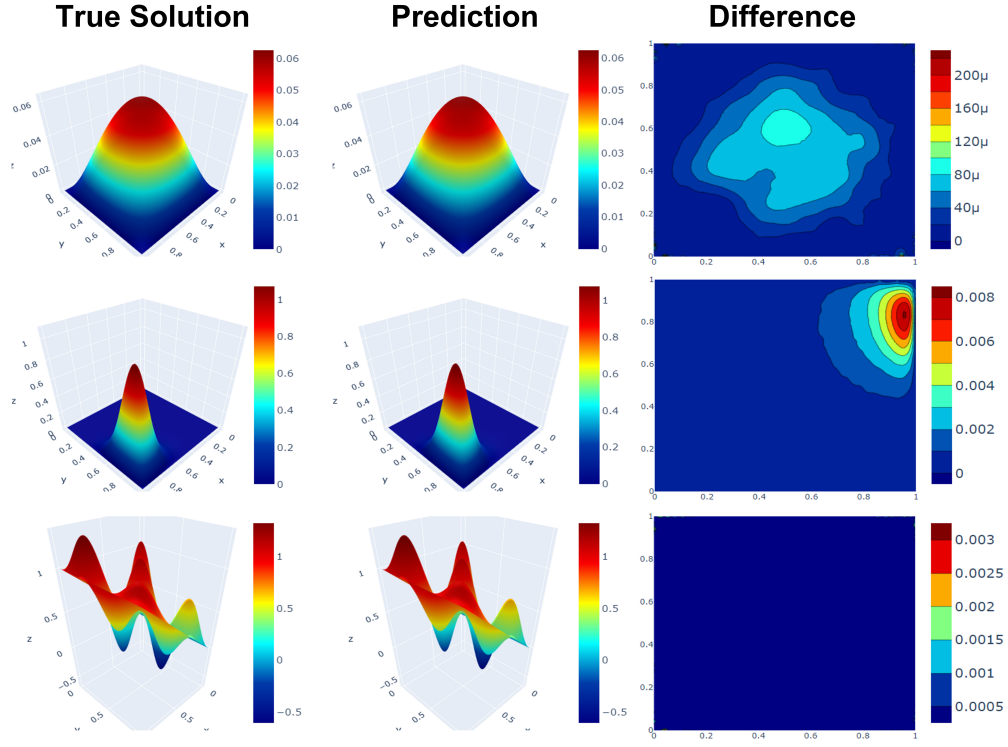
layer) at the coarser grids in such networks would be smaller than the size of the convolutional kernels in the network.

The results of Algorithm 1 for the non-constant diffusion problem defined in Section 2.4.1 are shown in Table 4. Here, we see more significant errors than with the constant diffusion problems. Figure 5 shows the true solution, predicted solution, and absolute difference for the non-constant diffusion problem on a 128×128 grid. This figure shows that the errors are mostly concentrated around the discontinuities in κ . We do not compare this to the finite difference method because the exact solution exhibits a singularity at the origin. Indeed, the gradient of the exact solution blows up at the point $(0, 0)$.

Because the finite difference solution solves the convex minimization problem (7), the finite difference approximation’s accuracy limits our approach’s accuracy. This explains the relatively poor performance in the case of discontinuous diffusion. It is well known that, due to the singularity at the origin, the exact solution to the discontinuous diffusion problem belongs to the Sobolev space $H^{1+\beta}(\Omega)$, where $\beta \approx 0.5354$ [31]. Hence, the finite difference method, which requires more regularity, performs poorly.

Problem	$\mathcal{M}$	$N = 32$		$N = 64$		$N = 128$	
		$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$
Bubble	500	4.7521e-05	5.2953e-04	4.8521e-05	8.2744e-04	4.2407e-04	1.6607e-03
	1,000	2.5330e-05	2.4341e-04	2.1550e-05	3.4875e-04	1.3244e-04	5.3431e-04
	2,000	1.4457e-05	1.4313e-04	9.1955e-06	1.4581e-04	4.0090e-05	3.0874e-04
	4,000	3.2266e-06	3.0962e-05	3.1159e-06	5.4674e-05	7.6852e-06	8.5074e-05
	8,000	2.2661e-06	2.9351e-05	3.7016e-06	4.9231e-05	3.7105e-06	4.4149e-05
Peak	500	2.0066e-02	1.2398e-01	1.0361e-02	1.2776e-01	3.2611e-02	2.8655e-01
	1,000	2.0603e-02	1.1937e-01	5.3352e-03	3.2162e-02	5.0469e-03	2.6498e-02
	2,000	2.0594e-02	1.1934e-01	5.5620e-03	3.2176e-02	1.3510e-03	8.0407e-03
	4,000	2.1275e-02	1.1932e-01	5.6542e-03	3.2192e-02	1.3851e-03	8.0365e-03
	8,000	2.0616e-02	1.1933e-01	5.5843e-03	3.2196e-02	1.4042e-03	8.0850e-03
Exp-Trig	500	2.8224e-03	1.8145e-02	1.0084e-03	1.5877e-02	2.6316e-03	3.0236e-02
	1,000	2.4582e-03	7.8004e-03	6.3506e-04	4.8903e-03	4.0585e-04	1.6215e-02
	2,000	2.3809e-03	7.8190e-03	5.9138e-04	1.8863e-03	1.8116e-04	3.6817e-03
	4,000	2.4604e-03	7.8169e-03	5.9671e-04	2.4966e-03	1.5110e-04	1.3024e-03
	8,000	2.3742e-03	7.8151e-03	5.8227e-04	1.8845e-03	1.4478e-04	5.7032e-04

Table 1: Accuracy of unsupervised predictions for a varying grid size N , number of optimization steps $\mathcal{M}$, and the depth of the U-Net set to three.



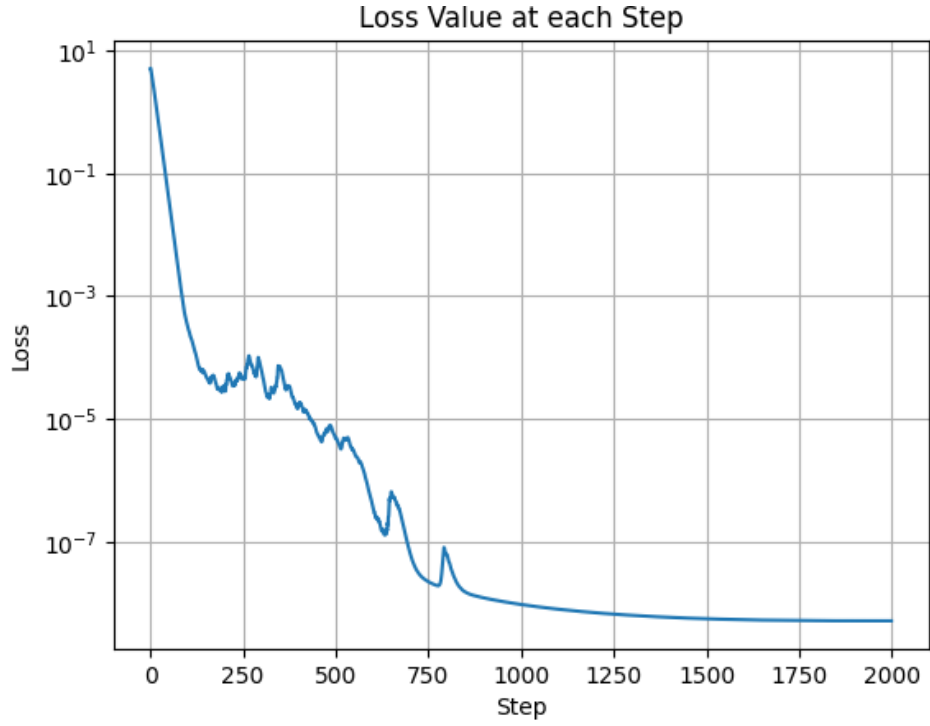


Figure 4: Loss values at each optimization step for the bubble function. Here, we use a network depth of three, a grid size of 128×128 , and 2,000 optimization steps.

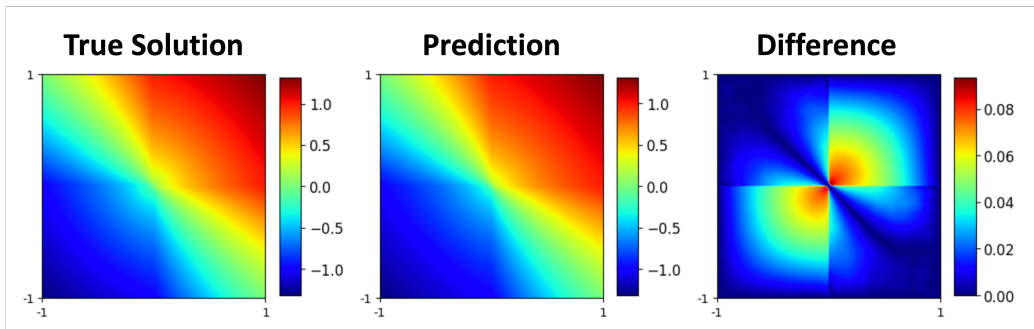


Figure 5: From left to right, contour plots of true solution, predicted solution, and absolute difference for the non-constant diffusion problem.

Problem	d	# Params.	$N = 32$		$N = 64$		$N = 128$	
			$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$
Bubble	2	283,713	3.2705e-06	3.8412e-05	2.8414e-06	4.1518e-05	5.4252e-04	1.9439e-03
	3	412,225	3.2266e-06	3.0962e-05	3.1159e-06	5.4674e-05	7.6852e-06	8.5074e-05
	4	541,889	-	-	2.6481e-06	4.1081e-05	3.2062e-06	6.0881e-05
	5	669,249	-	-	-	-	2.8731e-06	9.5620e-05
Peak	2	283,713	2.1274e-02	1.1932e-01	5.6116e-03	3.2174e-02	1.4157e-03	7.9525e-03
	3	412,225	2.1275e-02	1.1932e-01	5.6542e-03	3.2192e-02	1.3851e-03	8.0365e-03
	4	541,889	-	-	5.6673e-03	3.2192e-02	1.4087e-03	8.0848e-03
	5	669,249	-	-	-	-	1.4042e-03	8.0918e-03
Exp-Trig	2	283,713	2.4689e-03	7.8191e-03	5.9526e-04	1.8814e-03	2.8053e-02	7.5356e-02
	3	412,225	2.4604e-03	7.8169e-03	5.9671e-04	2.4966e-03	1.5110e-04	1.3024e-03
	4	541,889	-	-	5.9509e-04	1.8855e-03	1.5166e-04	1.8679e-03
	5	669,249	-	-	-	-	1.4944e-04	1.2124e-03

Table 2: Accuracy of unsupervised predictions for varying grid sizes N and U-Net depths d . We set the number of optimization steps to 4,000.

Problem	$N = 32$		$N = 64$		$N = 128$	
	$\ u - u_h\ _{2,h}$	$\ u - u_h\ _\infty$	$\ u - u_h\ _{2,h}$	$\ u - u_h\ _\infty$	$\ u - u_h\ _{2,h}$	$\ u - u_h\ _\infty$
Bubble	1.3582e-16	2.8449e-16	2.4524e-16	6.8695e-16	8.1134e-16	1.9776e-15
Peak	1.7597e-02	1.1151e-01	5.3441e-03	3.1282e-02	1.3988e-03	7.9696e-03
Exp-Trig	2.2986e-03	7.3266e-03	5.7262e-04	1.8251e-03	1.4303e-04	4.5588e-04

Table 3: Finite difference errors for selected elliptic problems for comparison.

Problem	$\mathcal{M}$	$N = 32$		$N = 64$		$N = 128$	
		$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$
Non. Const. Diff.	500	4.6597e-02	1.8380e-01	8.6890e-02	3.7069e-01	5.5651e-01	2.1105e+00
	1,000	3.9795e-02	1.6035e-01	4.9640e-02	1.3376e-01	2.0766e-01	6.4809e-01
	2,000	3.9718e-02	1.5799e-01	3.1458e-02	1.2455e-01	8.7473e-02	2.8797e-01
	4,000	3.9713e-02	1.5794e-01	3.1224e-02	1.2258e-01	4.9868e-02	1.1281e-01
	8,000	3.9710e-02	1.5800e-01	3.1045e-02	1.2221e-01	3.1002e-02	9.3761e-02

Table 4: Accuracy of unsupervised predictions on the non-constant diffusion problem for a varying grid sizes N , number of optimization steps $\mathcal{M}$, and the depth of the U-Net set to three.

Using the methods described in Section 2 and Algorithm 2, we approximate the finite difference solution to the parabolic problems defined in Section 2.4.2. The time step is $\tau = 0.1$. Except for the first time step, we set the number of optimization steps per time step to 250. Because we start with randomly initialized weights in the first time step, we set the number of optimization steps to 1,000 in the first time step. Table 6 shows the accuracy

Problem	d	# Params.	$N = 32$		$N = 64$		$N = 128$	
			$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$
Non. Const. Diff.	2	283,713	3.9709e-02	1.5798e-01	3.1294e-02	1.2265e-01	3.3859e-02	1.4377e-01
	3	412,225	3.9713e-02	1.5794e-01	3.1224e-02	1.2258e-01	4.9868e-02	1.1281e-01
	4	541,889	-	-	3.1110e-02	1.2282e-01	3.8937e-02	1.0907e-01
	5	669,249	-	-	-	-	3.6566e-02	1.0511e-01

Table 5: Accuracy of unsupervised predictions on non-constant diffusion problem for varying grid sizes N and U-Net depths d . We set the number of optimization steps to 4,000.

of our unsupervised predictions for varying grid sizes. For reference, Table 7 shows the accuracy of the finite difference method with backward Euler for varying grid sizes. Here, we see that Algorithm 2 achieves comparable accuracy to the finite difference method with backward Euler. However, we do not see the same convergence as with the finite difference approach for the trigonometric functions. The small number of optimization steps can explain this lack of convergence and, again, the weakness of first-order optimizers like Adam. Figures 6 through 8 show the true solution, predicted solution, and absolute difference for the first ten time steps of each problem. These figures show that our unsupervised approach produces visually accurate solutions. Finally, Table 8 displays the errors for several choices of activation functions at time $t = 0.5$. The errors are similar for ReLU, Tanh, Swish [15], and identity activation functions. These results indicate that linear activation functions (i.e., the identity) are sufficient for learning solutions to time-dependent problems.

Algorithm 1 may benefit from transfer learning (i.e., reusing weights from previous problems) with a supervised counterpart that is trained on a large labeled dataset. We randomly initialize the neural network weights in Algorithms 1 and 2 (first time step only). This random initialization may result in predictions with high errors at the beginning of training. Without optimal step sizes in our optimizer, these poor solutions may result in slow convergence towards an optimal solution. Using transfer learning with a pre-trained network that is trained in a supervised setting (i.e., with labeled training data), the initial predictions from our algorithms may be already close to optimal, resulting in faster convergence. One could view the use of these pre-trained networks as a sort of preconditioner for our neural network-based approach. Indeed, we see the benefits of transfer learning with Algorithm 2 after the first time step. Instead of reinitializing the weights for the subsequent time steps, we reuse the weights from the previous solution.

Problem	t	$N = 32$		$N = 64$		$N = 128$	
		$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$
Trig. ($n = 1$)	0.5	7.6825e-04	1.5388e-03	1.0661e-03	2.1703e-03	1.1562e-03	2.4109e-03
	1.0	5.2059e-04	1.0459e-03	7.0987e-04	1.4189e-03	7.7541e-04	1.5364e-03
	2.5	6.1710e-04	1.2354e-03	8.6825e-04	1.7483e-03	9.4440e-04	1.9382e-03
	5.0	1.6629e-04	3.3075e-04	2.5692e-04	5.2649e-04	3.0442e-04	6.2141e-04
Trig. ($n = 4$)	0.5	6.0011e-03	1.1977e-02	1.3953e-03	2.9254e-03	3.2371e-04	9.0033e-04
	1.0	3.6984e-03	7.3841e-03	8.5877e-04	1.8117e-03	1.9062e-04	5.3841e-04
	2.5	5.4592e-03	1.0897e-02	1.3062e-03	3.1186e-03	8.4891e-04	2.8031e-03
	5.0	1.9352e-03	3.8744e-03	4.8639e-04	1.3043e-03	9.8322e-04	2.5854e-03
Gaussian	0.5	3.0932e-03	3.8826e-02	7.1997e-04	1.0341e-02	1.8902e-04	2.5298e-03
	1.0	1.9069e-03	2.3921e-02	4.4389e-04	6.3892e-03	1.3421e-04	1.4701e-03
	2.5	2.8197e-03	3.5385e-02	6.5362e-04	9.5078e-03	2.7103e-04	2.3168e-03
	5.0	1.0004e-03	1.2579e-02	2.8684e-04	2.6686e-03	3.1119e-04	9.0614e-04

Table 6: Accuracy of unsupervised predictions for varying grid sizes and time steps for parabolic problems. The depth of the network is set to three, the number of optimization iterations at each time step to 250, and the time step to 0.1.

Problem	t	$N = 32$		$N = 64$		$N = 128$	
		$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _{\infty}$
Trig. ($n = 1$)	0.5	7.6693e-04	1.5299e-03	1.0560e-03	2.1106e-03	1.1255e-03	2.2507e-03
	1.0	5.2033e-04	1.0380e-03	7.0797e-04	1.4151e-03	7.5314e-04	1.5060e-03
	2.5	6.1623e-04	1.2293e-03	8.6455e-04	1.7280e-03	9.2434e-04	1.8484e-03
	5.0	1.5534e-04	3.0988e-04	2.3116e-04	4.6204e-04	2.4942e-04	4.9877e-04
Trig. ($n = 4$)	0.5	5.9961e-03	1.1961e-02	1.3890e-03	2.7763e-03	2.8791e-04	5.7573e-04
	1.0	3.7021e-03	7.3852e-03	8.5647e-04	1.7119e-03	1.7638e-04	3.5271e-04
	2.5	5.4540e-03	1.0880e-02	1.2656e-03	2.5296e-03	2.6449e-04	5.2889e-04
	5.0	2.8215e-03	5.6286e-03	6.5597e-04	1.3111e-03	1.3834e-04	2.7664e-04
Gaussian	0.5	3.0933e-03	3.8814e-02	7.1809e-04	1.0412e-02	1.8424e-04	2.4368e-03
	1.0	1.9068e-03	2.3920e-02	4.4286e-04	6.4085e-03	1.1490e-04	1.4920e-03
	2.5	2.8194e-03	3.5387e-02	6.5422e-04	9.5075e-03	1.6586e-04	2.2389e-03
	5.0	1.4619e-03	1.8355e-02	3.3909e-04	4.9403e-03	8.4882e-05	1.1716e-03

Table 7: Finite differences with backward Euler errors on selected parabolic problems for comparison. Here, the time step is 0.1.

That allows us to use a small number of optimization steps (i.e., $\mathcal{M} = 250$) at each time for parabolic problems.

It is important to note that our methods for estimating the finite difference solutions do not explicitly construct the finite difference matrix. Instead,

Problem	Activation	$N = 32$		$N = 64$		$N = 128$	
		$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$	$\ u - \hat{u}\ _{2,h}$	$\ u - \hat{u}\ _\infty$
Trig. ($n = 1$)	ReLU	7.6775e-04	1.5420e-03	1.0746e-03	2.1598e-03	1.2400e-03	2.5946e-03
	Tanh	7.6743e-04	1.5391e-03	1.0590e-03	2.1269e-03	1.1478e-03	2.3299e-03
	Swish	7.6905e-04	1.5538e-03	1.0623e-03	2.0923e-03	1.1637e-03	2.4307e-03
	Identity	7.6825e-04	1.5388e-03	1.0661e-03	2.1703e-03	1.1562e-03	2.4109e-03
Trig. ($n = 4$)	ReLU	5.9967e-03	1.1968e-02	1.5607e-03	4.1957e-03	9.2973e-04	2.7800e-03
	Tanh	5.9970e-03	1.1968e-02	1.3906e-03	2.8266e-03	3.1624e-04	8.8251e-04
	Swish	6.0061e-03	1.1990e-02	1.6013e-03	3.9862e-03	4.7380e-04	1.4040e-03
	Identity	6.0011e-03	1.1977e-02	1.3953e-03	2.9254e-03	3.2371e-04	9.0033e-04
Gaussian	ReLU	3.0932e-03	3.8828e-02	7.3665e-04	1.0873e-02	3.8312e-04	3.3270e-03
	Tanh	3.0930e-03	3.8808e-02	7.1797e-04	1.0419e-02	1.9917e-04	2.4221e-03
	Swish	3.0933e-03	3.8830e-02	7.4640e-04	1.0099e-02	6.0038e-04	4.4619e-03
	Identity	3.0932e-03	3.8826e-02	7.1997e-04	1.0341e-02	1.8902e-04	2.5298e-03

Table 8: Accuracy of unsupervised predictions for varying grid sizes and activation functions at time 0.5. The depth of the network is set to three, the number of optimization steps to 250, and the time step is 0.1.

we use our neural networks to map the source term f to the approximate finite difference solution. In this sense, we are learning the inverse mapping of the finite difference matrix. From a numerical point of view, this provides the advantage of not having to construct or store a finite difference matrix. Instead, we implement the required stencil(s) for our problem in the loss function and apply them appropriately. Only having to implement the stencils and not construct a finite difference matrix is also an advantage from an implementation point-of-view.

The justification for the lack of non-linear activation functions in our network is that the relationship between the source term f and the finite differences solution u_h is given by a system of linear equations of the form $Au_h = f$. In other words, we know a priori that our goal is to learn a linear relationship between f and our estimated solution. In many machine learning applications, the exact nature of the input and output relationship is unknown and assumed to be highly non-linear. Hence, including non-linear activation functions results in a network that learns a non-linear relationship between inputs and outputs. Using the identity function as an activation puts us outside the scope of the universal approximation theorem [34, 35, 36]. However, in our case, we do not need our neural network to be a universal approximator. The role of the network is to learn a linear relationship for a single example. Inputs other than f in Algorithms 1 and 2, like random

noise or constants, would break our assumption that the relationship between the network input and output is linear. Hence, a much larger (i.e., more parameters) non-linear network would be needed to learn such a relationship.

We utilize the PocketNet approach proposed by [22] in our proposed algorithms. This approach takes advantage of the similarity between the U-Net architecture and geometric multigrid methods to drastically reduce the number of parameters, while maintaining the same accuracy as conventional CNNs for medical imaging and scientific machine learning tasks [37, 38, 39]. Additionally, we replace transposed convolution with bilinear upsampling. We find that these changes save time and memory and yield the same accuracy that we see using conventional CNNs (i.e., doubling the number of channels at every depth). These results indicate that smaller neural networks (in terms of parameters) can achieve high accuracy for scientific machine learning tasks.

We see in Tables 2 and 5 that the depth of the U-Net architecture does not generally have a significant effect on our results. This indicates that, regardless of depth, the U-Net architecture is sufficiently expressive to learn a mapping from the right hand side f to an approximation of the finite difference solution u_h . However, non-U-shaped architectures like the HRNet may also produce similar or improved results [40]. Additionally, the use of residual or dense connections within the convolutions of our architecture may also be beneficial [41, 42]. Such block designs have been shown to speed up convergence to lower loss values for neural networks [43]. Finally, modifying our existing architecture by adding deep supervision could also speed up convergence to lower loss values in fewer iterations [44, 45].

The use of the weighting parameter α in (8) and (19) is necessary to enforce the given boundary conditions. We arrived at our proposed values of α via a grid search over a range of possible values. However, finding optimal values of weighting parameters like α is an open question [16]. Strongly enforcing the Dirichlet boundary conditions by modifying the network output to the prescribed values on the boundary is another approach that has been shown to be effective from an experimental and theoretical perspective [46, 24, 47]. However, both approaches of weakly and strongly enforcing boundary conditions in neural network-based methods are popular and have their own strengths and weaknesses [16]. In our case, strongly enforcing boundary conditions does not have a significant impact on the accuracy of our predicted solutions. Indeed, even with our weakly enforced approach, the boundary is not a significant source of error.

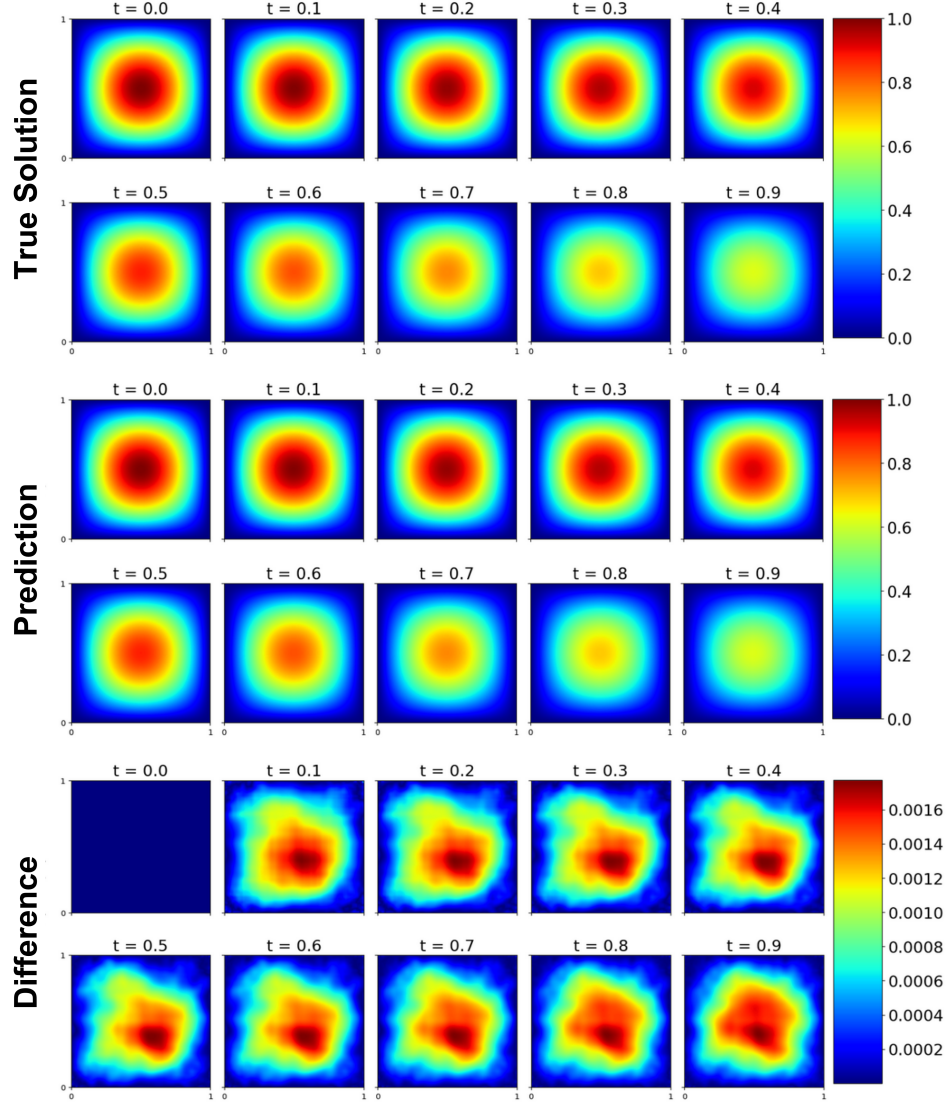


Figure 6: True solution (top), predicted solution (middle) and absolute difference (bottom) for the first ten time steps of the trigonometric problem presented in Section 2.4 with $n = 1$.

5. Conclusions

The results presented above show the effectiveness of our proposed unsupervised approaches for estimating the finite difference solution to elliptic

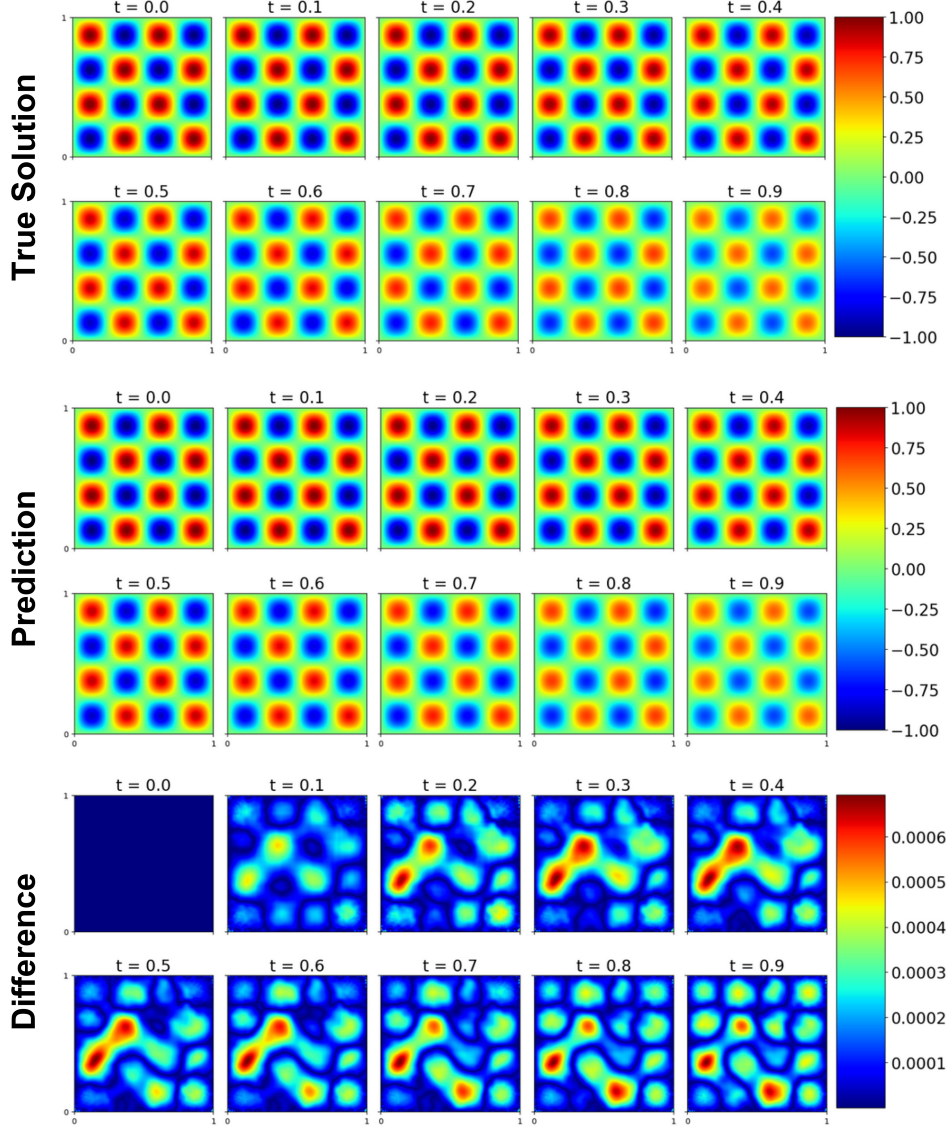


Figure 7: True solution (top), predicted solution (middle) and absolute difference (bottom) for the first ten time steps of the trigonometric problem presented in Section 2.4 with $n = 4$.

and parabolic problems. Unlike classical PINNs, our approach is influenced by numerical PDEs (i.e., the finite difference method), resulting in more explainable solutions. Additionally, unlike other CNN-based approaches, we

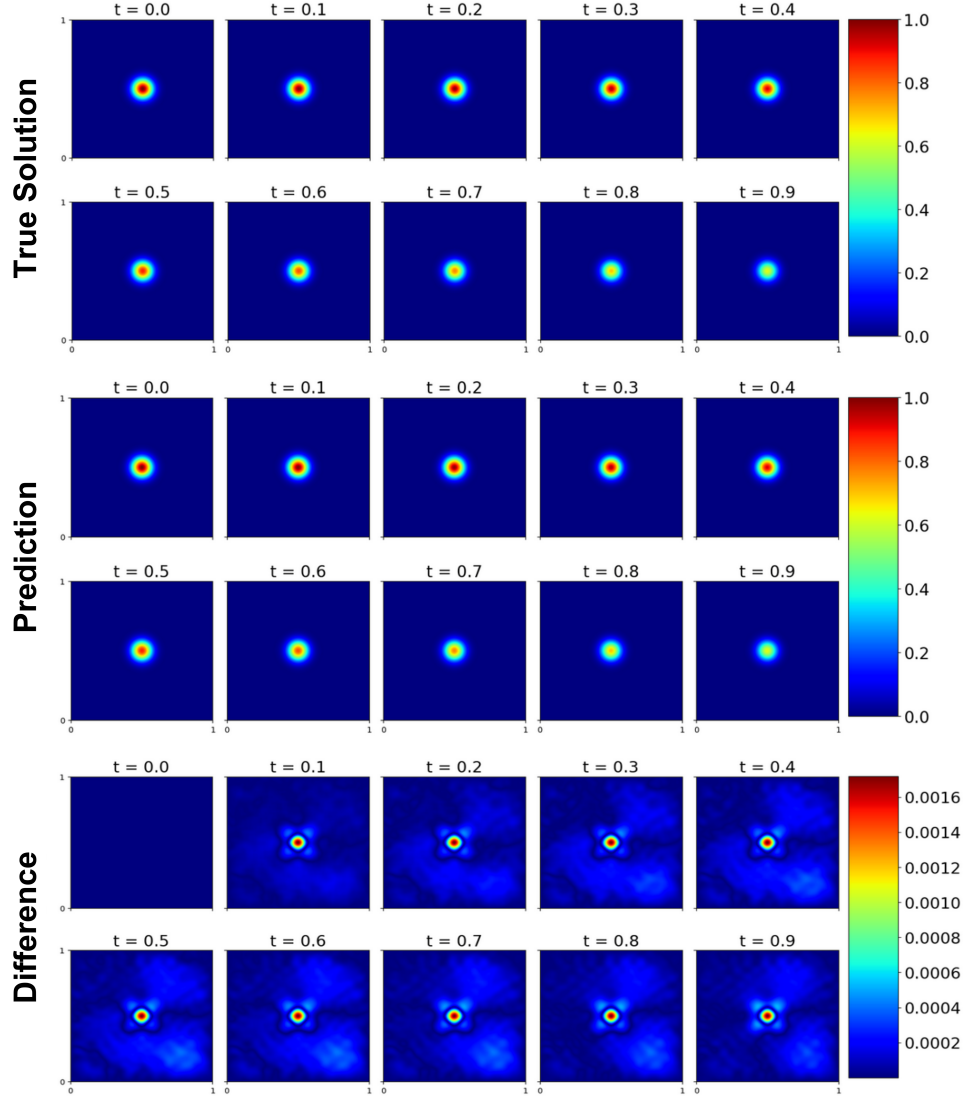


Figure 8: True solution (top), predicted solution (middle) and absolute difference (bottom) for the first ten time steps of the Gaussian problem presented in Section 2.4.

define the method for elliptic PDEs with non-constant diffusion coefficients and extend it to time-dependent problems. Finally, we use small linear CNNs, making our method computationally efficient.

Our approach could benefit finite difference solvers by producing better

initial guesses and/or acting as a preconditioner. With a few iterations, Algorithm 1 can produce good initial guesses for iterative solvers, thereby reducing the number of iterations required to solve the linear system resulting from (2) or (9). This same idea can also apply to time-dependent problems, but with initial guesses being produced at each time step. Additionally, because we employ identity for activation functions, our neural networks are linear. Hence, it may be possible to represent a pretrained architecture as a matrix. This resulting matrix could then be used as a preconditioner for finite difference solvers. This use of preconditioners and further testing on other kinds of PDEs, like convection-diffusion, coupled, and nonlinear problems, will be the object of future work.

6. Acknowledgements

The Department of Defense supports Adrian Celaya through the National Defense Science & Engineering Graduate Fellowship Program. Keegan Kirk is supported by the Natural Sciences and Engineering Research Council of Canada through the Postdoctoral Fellowship Program (PDF-568008). David Fuentes is partially supported by R21CA249373. Beatrice Riviere is partially supported by NSF-DMS2111459. This research was partially supported by the Tumor Measurement Initiative through the MD Anderson Strategic Research Initiative Development (STRIDE), NSF-2111147, and NSF-2111459.

References

- [1] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural Networks, vol. 2, pp. 359–366, 1989.
- [2] K.-I. Funahashi, “On the approximate realization of continuous mappings by neural networks,” Neural Networks, vol. 2, pp. 183–192, 1989.
- [3] M. Dissanayake and N. Phan-Thien, “Neural-network-based approximations for solving partial differential equations,” communications in Numerical Methods in Engineering, vol. 10, no. 3, pp. 195–201, 1994.
- [4] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” IEEE transactions on neural networks, vol. 9, no. 5, pp. 987–1000, 1998.

- [5] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,” arXiv preprint arXiv:1711.10561, 2017.
- [6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,” arXiv preprint arXiv:1711.10561, 2017.
- [7] J. Hou, Y. Li, and S. Ying, “Enhancing pinns for solving pdes via adaptive collocation point movement and adaptive loss weighting,” Nonlinear Dynamics, vol. 111, no. 16, pp. 15233–15261, 2023.
- [8] X. Zhang, B. Mao, Y. Che, J. Kang, M. Luo, A. Qiao, Y. Liu, H. Anzai, M. Ohta, Y. Guo, et al., “Physics-informed neural networks (pinns) for 4d hemodynamics prediction: An investigation of optimal framework based on vascular morphology,” Computers in Biology and Medicine, vol. 164, p. 107287, 2023.
- [9] M. Sarabian, H. Babaee, and K. Laksari, “Physics-informed neural networks for brain hemodynamic predictions using medical imaging,” IEEE transactions on medical imaging, vol. 41, no. 9, pp. 2285–2303, 2022.
- [10] D. Li, K. Deng, D. Zhang, Y. Liu, H. Leng, F. Yin, K. Ren, and J. Song, “Lpt-qpn: A lightweight physics-informed transformer for quantitative precipitation nowcasting,” IEEE Transactions on Geoscience and Remote Sensing, 2023.
- [11] S. Mowlavi and S. Nabi, “Optimal control of pdes using physics-informed neural networks,” Journal of Computational Physics, vol. 473, p. 111731, 2023.
- [12] F. Fotiadis and K. G. Vamvoudakis, “A physics-informed neural networks framework to solve the infinite-horizon optimal control problem,” in 2023 62nd IEEE Conference on Decision and Control (CDC), pp. 6014–6019, IEEE, 2023.
- [13] P.-H. Chiu, J. C. Wong, C. Ooi, M. H. Dao, and Y.-S. Ong, “Can-pinn: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method,” Computer Methods in Applied Mechanics and Engineering, vol. 395, p. 114909, 2022.

- [14] K. Tang, X. Wan, and C. Yang, “Das-pinns: A deep adaptive sampling method for solving high-dimensional partial differential equations,” Journal of Computational Physics, vol. 476, p. 111868, 2023.
- [15] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” arXiv preprint arXiv:1710.05941, 2017.
- [16] 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,” Journal of Scientific Computing, vol. 92, no. 3, p. 88, 2022.
- [17] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” Nature Reviews Physics, vol. 3, no. 6, pp. 422–440, 2021.
- [18] A. Bonfanti, R. Santana, M. Ellero, and B. Gholami, “On the generalization of pinns outside the training domain and the hyperparameters influencing it,” arXiv preprint arXiv:2302.07557, 2023.
- [19] K. Sirignano, J. and Spiliopoulos, “DGM: A deep learning algorithm for solving partial differential equations,” Journal of Computational Physics, vol. 375, pp. 1339–1364, 2018.
- [20] E. Wang and B. Yu, “The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems,” Communications in Mathematical Sciences, vol. 6, pp. 1–12, 2018.
- [21] J. Müller and M. Zeinhofer, “Error estimate for the Deep Ritz method with boundary penalty,” Machine Learning Research, vol. 145, pp. 1–20, 2022.
- [22] A. Celaya, J. A. Actor, R. Muthusivarajan, E. Gates, C. Chung, D. Schellingerhout, B. Riviere, and D. Fuentes, “Pocketnet: A smaller neural network for medical image analysis,” IEEE Transactions on Medical Imaging, vol. 42, no. 4, pp. 1172–1184, 2022.
- [23] K. L. Lim, R. Dutta, and M. Rotaru, “Physics informed neural network using finite difference method,” 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2022.

- [24] X. Zhao, Z. Gong, Y. Zhang, W. Yao, and X. Chen, “Physics-informed convolutional neural networks for temperature field prediction of heat source without labeled data,” Engineering Applications of Artificial Intelligence, vol. 117, p. 105516, 2023.
- [25] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pp. 234–241, Springer, 2015.
- [26] A. Celaya, B. Denel, Y. Sun, M. Araya-Polo, and A. Price, “Inversion of time-lapse surface gravity data for detection of 3-D CO₂ plumes via deep learning,” IEEE Transactions on Geoscience and Remote Sensing, vol. 61, pp. 1–11, 2023.
- [27] X. Yang, X. Chen, and M. M. Smith, “Deep learning inversion of gravity data for detection of CO₂ plumes in overlying aquifers,” Journal of Applied Geophysics, vol. 196, p. 104507, 2022.
- [28] Y. Sun, B. Denel, N. Daril, L. Evano, P. Williamson, and M. Araya-Polo, “Deep learning joint inversion of seismic and electromagnetic data for salt reconstruction,” SEG Technical Program Expanded Abstracts 2020, pp. 550–554, 2020.
- [29] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-Net: learning dense volumetric segmentation from sparse annotation,” in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19, pp. 424–432, Springer, 2016.
- [30] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein, “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation,” Nature Methods 2020 18:2, vol. 18, pp. 203–211, 12 2020.
- [31] B. Riviere and M. F. Wheeler, “A posteriori error estimates for a discontinuous Galerkin method applied to elliptic problems. log number: R74,”

- Computers & Mathematics with Applications, vol. 46, no. 1, pp. 141–163, 2003.
- [32] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.
 - [33] F. Chollet et al., “Keras.” <https://keras.io>, 2015.
 - [34] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.
 - [35] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of control, signals and systems, vol. 2, no. 4, pp. 303–314, 1989.
 - [36] K.-I. Funahashi, “On the approximate realization of continuous mappings by neural networks,” Neural networks, vol. 2, no. 3, pp. 183–192, 1989.
 - [37] H. Zhang, M. Liu, Y. Qi, Y. Ning, S. Hu, L. Nie, and W. Zhang, “Efficient brain tumor segmentation with lightweight separable spatial convolutional network,” ACM Transactions on Multimedia Computing, Communications and Applications, 2024.
 - [38] A. Celaya, D. Fuentes, and B. Riviere, “Fmg-net and w-net: Multigrid inspired deep learning architectures for medical imaging segmentation,” in Neural Information Processing Systems Conference: LatinX in AI (LXAI) Research Workshop 2023, LXAI, 2023.
 - [39] A. Celaya, B. Denel, Y. Sun, and M. Araya-Polo, “Inversion of time-lapse surface gravity data for monitoring of 3d co2 plumes via physics informed neural networks,” in Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing (PP), pp. 1–12, SIAM, 2024.
 - [40] K. Sun, B. Xiao, D. Liu, and J. Wang, “Deep high-resolution representation learning for human pose estimation,” in CVPR, 2019.
 - [41] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in European conference on computer vision, pp. 630–645, Springer, 2016.

- [42] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708, 2017.
- [43] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the loss landscape of neural nets,” Advances in neural information processing systems, vol. 31, 2018.
- [44] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, “Deeply-supervised nets,” in Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (G. Lebanon and S. V. N. Vishwanathan, eds.), vol. 38 of Proceedings of Machine Learning Research, (San Diego, California, USA), pp. 562–570, PMLR, 09–12 May 2015.
- [45] R. Li, X. Wang, G. Huang, W. Yang, K. Zhang, X. Gu, S. N. Tran, S. Garg, J. Alty, and Q. Bai, “A comprehensive review on deep supervision: Theories and applications,” arXiv preprint arXiv:2207.02376, 2022.
- [46] S. Berrone, C. Canuto, M. Pintore, and N. Sukumar, “Enforcing dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks,” Heliyon, vol. 9, no. 8, 2023.
- [47] M. Zeinhofer, R. Masri, and K.-A. Mardal, “A unified framework for the error analysis of physics-informed neural networks,” arXiv preprint arXiv:2311.00529, 2023.