POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Surrogate Models for Parametric PDEs via Graph-Informed Neural Networks

Supervisors

Dott. Francesco Della Santa

Dott. Maria Strazzullo

Candidate

Susanna Olivero

Academic year 2023-2024

Table of Contents

Mathematics is not about numbers,
equations, computations, or algorithms:
it is about understanding

- William Paul Thurston

Abstract

This thesis focuses on training surrogate models based on Deep Learning (DL) to predict the solutions of parametric Partial Differential Equation (PDE) problems. In particular, the PDE problems we consider have most of their parameters used to characterize the boundary conditions, not only the physical properties embedded in the differential equations. The scenarios examined involve two types of problems, one purely diffusion-based and one involving both diffusion and convection. Several kinds of DL models are taken into account, including a novel spatial-based graph network called Graph-Informed Neural Network (GINN). Error statistics are computed to understand how the models' predictions are affected by the model architecture, the amount of training data, the hyperparameters of the network, and the physical parameters of the problem. The experiments demonstrate the effectiveness of the GINNs as surrogate models for parametric PDEs, also compared to more traditional DL models.

1 Introduction

1.1 Motivation

Parametric Partial Differential Equations (PDEs) are fundamental to modeling a variety of physical, biological, and engineering systems, from fluid dynamics and weather forecasting to material science and quantum mechanics. Parametric problems are those in which the system's behavior depends on specific input parameters, such as material properties, boundary conditions, or external forces. As the parameters vary, the solution to the PDE changes, making these problems essential for understanding how systems respond to different conditions.

Traditional methods for solving PDEs, such as finite element or finite difference methods, often require a fine discretization of the spatial domain. This leads to a computationally expensive process, especially when the dimension of the parameter space is large or when the PDE domain is characterized by a high geometrical complexity; therefore, finding solutions in real-time or over large parameter ranges is a challenging task. Furthermore, in many cases, obtaining accurate solutions to PDEs requires solving the problem iteratively, which becomes impractical due to the enormous computational costs involved.

The difficulty of traditional approaches motivates the need for more efficient computational techniques, and this is where Deep Learning (DL) models come into play. DL models, particularly Neural Networks (NNs), have proven to be efficient and effective because they can learn complex, non-linear relationships between inputs and solutions, allowing them to generalize across a wide range of conditions [6]. Once trained, these models can provide rapid, real-time predictions without the need for repetitive and computationally expensive calculations, making them particularly suitable for high-dimensional and large-scale problems.
NNs offer a powerful and effective alternative to classical numerical methods, enabling rapid and accurate solutions to complex problems that are otherwise computationally prohibitive. The idea is that a dedicated NNs can be trained to learn the complex mappings between parameters and the corresponding PDE solutions, bypassing the need to repeatedly solve the equation from scratch [18].

1.2 Contribution

In this thesis, we specifically address two parametric problems. The first is a relatively classic problem in the field of parametric PDEs, involving pure diffusion where the parameters are mainly related to the boundary conditions (BCs). The second is a much more complex problem that incorporates both diffusion and convection, introducing additional physical parameters. Both of our problems fall within a particularly complicated family of parametric PDEs, mainly characterized by parametrized BCs.
The primary objective of this work is to find an efficient way to solve these problems using NNs. Indeed, to the best of our knowledge, this is the first time DL models are being applied to problems where the BCs are treated with a local structure. The surrogate model we aim to develop must be reliable, with low approximation errors, and capable of real-time consultation, which is why we are using NNs.
To achieve this, we trained and tested various DL models, including traditional Fully-Connected architectures, with and without residual blocks, as well as a novel spatial-based graph network called Graph-Informed Neural Network (GINN). Different training regimes were applied to these models, varying the amount of training data, the network hyperparameters, and the physical parameters of the problem.
By analyzing the results, we aim to understand the behavior of the solutions as the physical parameters of the problem, including the parametrized BCs, vary. As mentioned earlier, the introduction of parametrized BCs is one of the most innovative aspects of this work.
Given that GINNs represent an innovative NN, another contribution of this thesis is the study of their behavior in depth to enable their most effective use in future applications.

1.3 Structure

In this section we provide a brief overview of the entire thesis structure to offer a roadmap into the project. Following this introductory chapter, there are five additional chapters.
The second chapter is primarily theoretical and provides the foundations upon which the research is built. We begin with a general introduction to the NNs, with a particular attention on the Graph Neural Networks. We then proceed to present and explain the novel GINN, the one that we have used for our simulations.
The third chapter introduces the theoretical problem we aim to address, specifically parametric PDE problems, with a focus on parametrized boundary conditions (BCs). We begin with a general overview of PDEs, followed by the mathematical

formulation of the parametrized problem and a detailed presentation of the theoretical problem we want to solve. The chapter concludes by presenting the solving methods for PDEs, with particular attention to NNs.
The fourth chapter starts by presenting the actual physical problems that we aim to solve and then outlines the methodology adopted in the experiments. It elucidates the approach taken in conducting the research and its various phases, as data preparation, experiments setup and evaluation. Towards the end of the chapter, we also provide some information regarding implementation details, focusing on the practical aspects of the research. This chapter acts as a bridge, connecting the theoretical foundations established in the preceding chapters with the practical development of the research.
In the fifth chapter we present the results of our experiments. For the analysis, we mainly computed error statistics to understand how the models' predictions are affected by the model architecture, the amount of training data, the hyperparameters of the network, and the physical parameters of the problem.
In the final chapter we sum up the content of this thesis, providing a comprehensive conclusion. Additionally, we look ahead to explore possible areas for future research and development.

2 Neural Networks

Neural Networks (NNs) are a class of Machine Learning (ML) models originally inspired by the structure and functioning of the human brain. They consist of interconnected layers of units, or neurons, each of which processes and transmits information. These models have seen three waves of popularity over the past 80 years [35] and we present them briefly.

The first wave, *Cybernetics* (1940s-1960s), introduced the first artificial neuron model and early NNs like the Perceptron [34], which laid the groundwork for modern DL, and ADALINE [42], which pioneered the use of gradient-based stochastic optimization techniques. Although they demonstrated highly interesting properties, they faced important limitations such as the inability to learn simple functions, such as the logic relation XOR [35].
The second wave, *Connectionism* (1980s-1990s), revived interest in NNs with the backpropagation algorithm and the concept of distributed representation. The core concept of connectionism, inspired by biology, is that a multitude of simple computational units, such as artificial neurons, can collectively mimic "intelligent behavior" when interconnected [35]. This wave of interest in NNs research came to an end in the mid-1990s. During this time, other areas of ML, such as kernel machines, progressed showing favorable outcomes, while NNs did not fulfill the expectations.
With advancements in computational power and the availability of large datasets, it has become possible to generate deeper architectures, creating the famous ML's sub-field, named *Deep Learning* (DL). This marks the third wave, which began in 2006 and has led to models that are now a dominant force in the field of artificial intelligence. These architectures can capture intricate patterns and complex relationships within data through the composition of highly non-linear parametric functions, resolving tasks that were previously considered challenging or even intractable [35].

After this brief historical introduction on the evolution of NNs, we present the structure of the chapter. In the first section, we illustrate some of the classic and well-known architectures, including the ones used for our simulations. In the second section we focus on Graph NNs and, in the last section, we present the novel spatial-based graph network we have used, the Graph-Informed Neural Network (GINN).

2.1 Classic and Well-known Neural Networks

NNs can be categorized into various types based on their architecture and use cases and, in this section, we introduce the most commonly used categories according to the literature. Additionally, we will focus on the mathematical nature of these models, presenting the characterizing layers.

Fully-Connected Neural Networks

Fully-Connected Neural Networks (FCNNs) are the simplest and oldest type of artificial NNs, also called Multi-Layer Perceptrons (MLPs) [29].
With this architecture the information flows in one direction, forward, from the input layer, through the hidden layers, and finally to the output layer, see Figure 2.1. Each neuron in a layer is fully connected to neurons in the previous layer; for this reason, such kind of layers are called Fully-Connected (FC) Layers, giving the name also to the NN model [23].

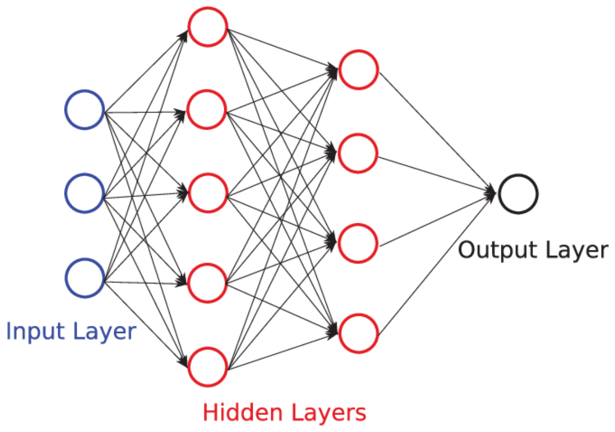


Figure 2.1: Architecture of a simple FCNN.

A FC layer (L^{FC}) with input dimension $c \in \mathbb{N}$ and output dimension $d \in \mathbb{N}$, defines its action as a function $\mathcal{L}^{FC} : \mathbb{R}^c \to \mathbb{R}^d$ such that:

$$\mathcal{L}^{FC}(\boldsymbol{x}) = \boldsymbol{\sigma}\left(W^T\boldsymbol{x} + \boldsymbol{b}\right), \quad \forall \boldsymbol{x} \in \mathbb{R}^c,$$

where

- $\boldsymbol{x} \in \mathbb{R}^c$ is the input;
- $W \in \mathbb{R}^{c \times d}$ is the weight matrix, where each component $w_{i,j}$ is the weight of the connection between the i-th unit of the previous layer and the j-th unit of the actual layer, for each $i = 1, ..., c$ and $j = 1, ..., d$;

- $\boldsymbol{b} \in \mathbb{R}^d$ is the bias vector;
- $\boldsymbol{\sigma}$ is the element-wise application of the layer's activation function $\sigma : \mathbb{R} \to \mathbb{R}$.

FCNNs are often used for classification and regression, tasks where both the inputs and the outputs are vectors (or are modelled as vectors). From a mathematical point of view, FCNNs are used for learning functions $F : \mathbb{R}^n \to \mathbb{R}^m$, where $m \geq 1$ and $n \geq 1$ or, typically, $n \gg 1$.
This type of architecture has been used for some of the experiments carried out as part of the thesis project.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of NN primarily used for processing structured grid data, such as images. They emerged during the third wave that characterizes NNs and represent the true breakthrough behind the great success of DL models. The key innovation of CNNs is their ability to handle non-vectorial inputs, such as matrices or tensors (e.g., images), indeed, before CNNs, it was necessary to vectorize all inputs that were not already vectors. An important property of this NN is the capability of reducing the number of weights, compared to a FC layer, and so increasing the depth of the NN [14].
In general, CNNs are designed to automatically and adaptively learn spatial hierarchies of features through the so called convolutional and pooling, besides the FC layers. Convolutional layers apply filters to the input data to create what are called feature maps or activation maps. These layers are usually followed by pooling layers that down-sample the data, reducing its dimensionality while preserving important features; the most used are max and average pooling. Typically, near the end of the architecture, FC layers are used for learning the tasks given the encoding of the precedent layers, see Figure 2.2.

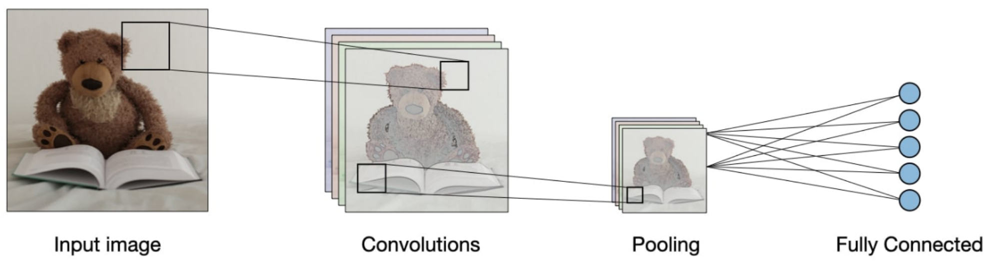


Figure 2.2: Schema of a simple CNN [1].

This type of NN is presented here not because it was actually used in our experiments, but because the convolutional layer served as an inspiration for the novel GINN, which we will introduce in the next section.

Residual Neural Networks

Residual Neural Networks (ResNets) are a specific type of NN characterized by the use of residual blocks of layers in their architecture [13].
The key feature is the so-called "residual connection", which adds the input of the subnetwork to its output, see Figure 2.3.
From a mathematical perspective, given a residual block of m layers and an input x, we denote the output after the m layers as $F(x)$, and the final output of the entire block as

$$y = F(x) + x,$$

where the sum represents the residual connection.

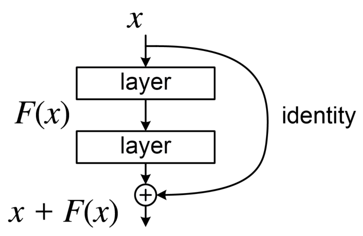


Figure 2.3: A residual block where the residual connection skips two layers [39].

This approach was developed to address the degradation problem (or vanishing gradient problem), where adding more hidden layers leads to the performance of a DL model to saturate and then declining. Residual blocks enable a deeper NN, allowing deeper layers to learn from the residual errors of lower layers. Additionally, these blocks help preserve information from earlier layers, enabling for better feature learning and improved model performance [35].
ResNet has proven effective in many real-world applications, such as Google's Transformer.

The ResNet architecture was used for some of the simulations carried out as part of the thesis project. In fact, one of the goals of our surrogate models for PDEs is both to create a deeper network and to preserve the initial features, as the boundary conditions and physical parameters are all established at the beginning.

2.2 Graph Neural Networks

Although DL excels at identifying hidden patterns in Euclidean data, there is a growing range of applications where data is represented as graphs. The complexity of graph data has posed significant challenges for existing ML algorithms since graphs can be very irregular. With variable sizes of unordered nodes and differing

numbers of neighbors for each node, certain key operations (such as convolutions) straightforward in the image domain, become difficult to implement in the graph domain. Additionally, a fundamental assumption of traditional ML algorithms is that instances are independent of one another. This assumption does not apply to graph data, where each instance (node) is connected to others through various types of links.
Recently, the NN community has made significant advancements in this field, extending DL techniques to graph-structured data through *Graph Neural Networks* (GNNs).

GNNs have their origins in the late 2000s, though they were initially hindered by high computational costs. Indeed, the first architectures fall into the category of *recurrent GNNs*, where the neighbor information are propagating in an iterative manner until a stable point is reached, generating a computational expensive process. However, the success of CNNs led to the development of a new generation of GNNs, which redefined the concept of convolutions for graph data and gave rise to *Graph Convolutional Networks* (GCNs) [5].
In addition to recurrent and convolutional GNNs, there are other types of GNNs, such as *graph autoencoders*, often used for unsupervised learning and graph reconstruction tasks, and *spatial–temporal GNNs*, tailored for modeling data that varies across both spatial and temporal dimensions. If the reader is interested, they may refer to the article [43].

2.2.1 Graph Convolutional Networks

GCNs extend the concept of convolution from grid-based data to graph-based data. Therefore, the approach consists in generating the representation of a node v by combining its own features x_v with the features x_u of its neighboring nodes $u \in N(v)$, where $N(v)$ is the set of neighbours of v in the graph. As classified in [43], GCNs fall into two main categories: *spectral-based* GCNs, which rely on spectral graph theory, and *spatial-based* GCNs, which aggregate information only from neighboring nodes [45].
Spectral-based methods are based on the mathematical principles of graph signal processing but, currently, spatial-based GCNs are favored in many applications due to their greater flexibility and efficiency. They are commonly used for various tasks on graph data, including: (i) semi-supervised node regression or classification, (ii) edge classification or link prediction, and (iii) graph classification [5]. On the other hand, the literature suggests that spectral-based GCNs performance tends to degrade as the number of graph convolutional layers increases [22].

Similar to the convolution operation of a traditional CNN applied to an image, spatial-based methods define graph convolutions based on the spatial relationships between nodes. An image can be viewed as a specific type of graph where each pixel acts as a node and is directly connected to its neighboring pixels, as illustrated in Figure 2.4(a). In this context, a filter is applied to a $n \times n$ patch by taking the weighted average of the pixel values of the central node and its surrounding neighbors across each channel. In a similar fashion, spatial-based graph convolutions combine the representation of a central node with the representations of its neighboring nodes to produce an updated representation for the central node, as shown in Figure 2.4(b).

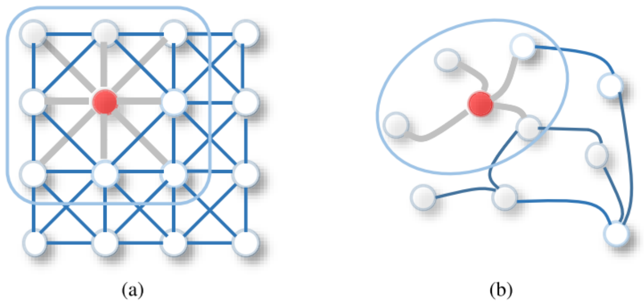


Figure 2.4: Example of a 2-D image convolution and a graph convolution [43].

The *Neural Network for Graphs* (NN4G) is one of the first spatial-based GCN introduced in the literature [25]. It carries out graph convolutions by directly summing the information from a node's neighbors and it also utilizes residual connections and skip connections to retain information across layers. More specifically, NN4G is based on the concept of *context window* which is defined for each state variable $x_i(v)$, given a vertex v. This context, $\mathcal{C}(x_i(v))$, refers to the collection of all state variables that, directly or indirectly, influence the determination of $x_i(v)$ [25]. Let $\mathcal{N}(v)$ be the set of the vertices adjacent to v, the context of $x_i(v)$ is expressed as

$$\mathcal{C}(x_i(v)) = \bigcup_{j=1}^{i-1} \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \mathcal{C}(x_j(u));$$

where $x_j(u)$ represents the direct contribution and $\mathcal{C}(x_j(u))$ the indirect one, which contribute to the determination of $x_j(u)$.
In the base case, $i = 2$, there is no contribution to the computation of $\mathcal{C}(x_1(u))$ so the formula results

$$\mathcal{C}(x_2(v)) = \bigcup_{u \in \mathcal{N}(v)} x_j(u) \cup \mathcal{C}(x_1(u)) = x_1\mathcal{N}(v),$$

having $\mathcal{C}(x_1(u)) = \emptyset$.
The two equations just presented are also known as the message passing equations for NNs.

Another type of spatial-based GCN is the *Diffusion Convolutional Neural Network* (DCNN) [15]. It considers graph convolutions as a diffusion process, assuming that information is passed from one node to its neighboring nodes based on a specific transition probability. This process continues until information distribution reaches equilibrium after multiple iterations.

Despite their effectiveness, GCNs face some challenges, notably: (i) constructing deep architectures that maintain strong performance, and (ii) achieving scalability for large graphs. So, even though the success of DL architectures lies in its depth, deeper GCNs do not necessarily yield better results [43].

2.3 Graph-Informed Neural Network

In this section, we present the Graph-Informed Neural Network (GINN), a new architecture that extends the basic formulation of spatial-based graph convolutional networks.

This novel architecture is specifically designed for regression tasks involving graph-structured data that are not well-suited to traditional GNNs. There are many applications of this type in interesting fields, such as network interdiction models, circulation with demand problems, and flux regression problems in underground fractured media [5]. The key point is that a standard MLP or its appropriate variants can effectively handle this regression task on graph data, as it implicitly learns node relationships during training, but existing GCNs are not as well-suited for this type of regression task compared to MLPs. Indeed, as mentioned in Section 2.2.1, GCNs are primarily designed for different types of tasks and often struggle to utilize deep architectures effectively.
The aim of the authors of the GINN [5] is to build a novel Spatial GCN architecture that is able to exploit its depth and the graph structure to enhance NN training, if compared to an MLP. Specifically, they introduced a new Graph-Informed (GI) layer that leverages the adjacency matrix of a given graph to determine the unit connections within the NN. The GINN architecture, therefore, consists of a series of these new GI layers, which will be presented in the next sub-section.
The convolution operation used for graph data in this architecture is more similar to CNN convolutions than to those in other GCNs. There are also similarities with DCNNs [15] and classic NN4G layers [25], in particular with the *message passing equations* presented in Section 2.2.1.

Numerical experiments presented in [5] have demonstrated the effectiveness of GI layers and their potential. Specifically, these experiments revealed that GINNs exhibit enhanced regression capabilities compared to MLPs, largely due to their ability to address the depth-related challenges commonly encountered in other

GCNs.
The authors also suggest that some graph classification tasks involving vertex labels can be addressed by simply adding a softmax layer at the end of the GINN, thereby extending the model.

2.3.1 The Graph-Informed Layers

In this section, we present, mainly from a mathematical perspective, the GI layer as defined in [5].

Given some graph-structured data, we have a graph G of n nodes, and we denote by V the set of the vertices and by E the set of the edges; i.e., $G = (V, E)$. The adjacency matrix of G is denoted as $A \in \mathbb{R}^{n \times n}$ and we define $\hat{A} := A + \mathbb{I}_n$, where $\mathbb{I}_n$ is the $n \times n$ identity matrix.
As said before, the GINN architecture is specifically designed for regression tasks, so, let's assume that the objective of this problem is to find the solution values only on a subset of m nodes, $\hat{V} \subset V$, where $\hat{V} \in \mathbb{R}^m$. In this case, a regression task on the graph G can be described by a function $F : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$. The function F depends on the adjacency matrix A of G.

Typically, the goal of a generic NN layer is to obtain the output feature of each unit by summing up some input features multiplied by some weights. The basic idea behind a GI layer (L^{GI}) is that we consider as input features only those of the node itself and its neighbors, each one multiplied by a weight assigned to the correspondent graph node. This is in contrast to the case of a classic FC layer (L^{FC}), where all features are considered. As mentioned before, the idea come from the structure of other GNNs, especially the ones that are based on the convolutional layers.
More precisely, the L^{GI} extends the functionality of convolutional layer filters to graph-structured data. The aim is to capture the implicit relationships between the features of neighboring graph nodes and to utilize the sparse interactions and parameter-sharing characteristics that are typical of CNNs. To do so, the convolution operation for the node v_i is re-define as

$$x_i' = \sigma \left(\sum_{j \in N_{in}(i) \cup \{i\}} x_j w_j + b_i \right),$$

where

- x_i' output feature of node v_i;
- x_j input features of node v_j, $j = 1, ...n$;
- w_j weight for node v_j, $j = 1, ...n$;

- $N_{in}(i)$ indicates the set of nodes v_j, such that there exists an incoming edge for the node v_i, $(v_j, v_i) \in E$;
- b_i is the bias corresponding to node v_i;
- $\sigma : \mathbb{R} \to \mathbb{R}$ is the layer's activation function.

As said before, to compute x'_i the layer act only on x_i and on the incoming neighbors. For a better understanding, refer to Figure 2.5.

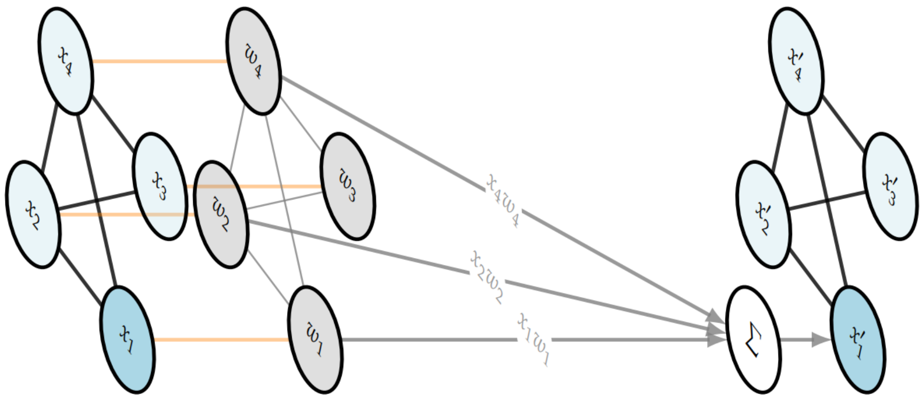


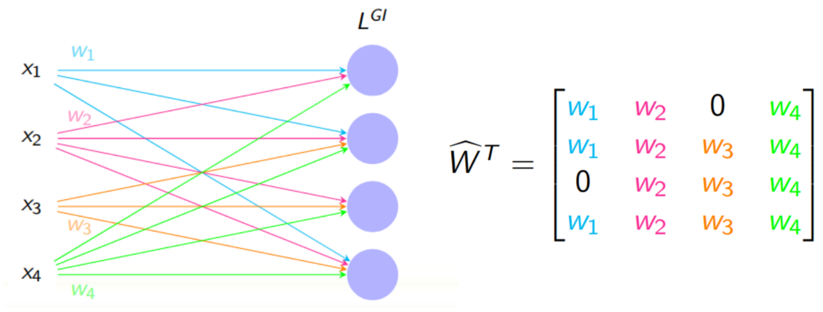
Figure 2.5: Example of the action of a filter on a graph with $n = 4$ nodes [5].

It is interesting to note that the L^{GI} can be seen as a constrained L^{FC} where the weights are designed so that

$$w_{ji} = \begin{cases} w_j, & \text{if} \quad (v_j, v_i) \in E \\ w_i, & \text{if} \quad j = i \\ 0, & \quad \text{otherwise} \end{cases}$$

for each $i, j = 1, ..., n$. See Figure 2.6.

Another characteristic of these new layers is the possibility to describe them through a compact mathematical formulation, which not only greatly simplifies understanding but also brings advantages in terms of implementation and computational efficiency. Moreover, the development of a compact expression enables us to generalize the model's process of progressively incorporating contextual information as new state variables are added [25]. We also point out to the reader that it is not a given that a layer of a NN can be described in a compact manner. Let us denote the L^{GI}'s action as the function $\mathcal{L}^{GI}$, and we are ready to present the L^{GI} formula.



L^{GI}
x_1
x_2
x_3
x_4
w_1
w_2
w_3
w_4
$\widehat{W}^T = \begin{bmatrix} w_1 & w_2 & 0 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ 0 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix}$

Figure 2.6: Example of how an L^{GI} can be seen as a constrained L^{FC} on a graph with $n = 4$ nodes as the one in Figure 2.5.

In the simplest case, i.e., the one with one feature per node for both input and output, the function $\mathcal{L}_1^{GI} : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$\mathcal{L}_1^{GI}(x) = \boldsymbol{f}\left(\widehat{W}^T x + b\right),$$

where

- $w \in \mathbb{R}^n$ is the weights vector associated with each vertex in V and $\widehat{W}$ is defined as
$$\widehat{W} := diag(w)\hat{A};$$
- $b \in \mathbb{R}^n$ is the biases vector;
- $\boldsymbol{f}$ is the element-wise application of the layer activation function $f : \mathbb{R} \to \mathbb{R}$.

The next step is to generalize the input features, enabling the layers to accept any arbitrary number $K \geq 1$ of input features from each node. In this case the function $\mathcal{L}_2^{GI} : \mathbb{R}^{n \times K} \to \mathbb{R}^n$ is defined as

$$\mathcal{L}_2^{GI}(X) = \boldsymbol{f}\left(\widetilde{W}^T \text{vertcat}(X) + b\right),$$

where

- $X \in \mathbb{R}^{n \times K}$ is the input matrix whose row $i \in \{1, ..., n\}$ describes the K features $x_{i1}, ..., x_{iK}$ of node v_i, and $\text{vertcat}(X) \in \mathbb{R}^{nK}$ is the vector obtained by concatenating the columns of X;
- $w_{\cdot 1}, ..., w_{\cdot K} \in \mathbb{R}^n$ are the weights vectors associated at the k-th input features for each vertex in V and $\widetilde{W}$ is defined as
$$\widetilde{W} := \begin{bmatrix} \widehat{W}^{(1)} \\ \vdots \\ \widehat{W}^{(K)} \end{bmatrix} = \begin{bmatrix} diag(w_{\cdot 1})\hat{A} \\ \vdots \\ diag(w_{\cdot K})\hat{A} \end{bmatrix} \in \mathbb{R}^{nK \times n}.$$

As before, the bias vector is added and then the activation function is applied. Moreover, we can notice the approach for building $\mathcal{L}_1^{GI}$ and $\mathcal{L}_2^{GI}$ is the same, indeed:

$$\sum_{k=1}^{K} \widehat{W}^{(k)T} x_{\cdot k} = \widetilde{W}^{T} \text{vertcat}(X)$$

It is important to highlight that the operations just described are an adaptation of convolutional layer operations for graph-based inputs. Specifically, the input $X \in \mathbb{R}^{n \times K}$ can be seen as an $n \times 1$ image with K channels, while $w_{\cdot k}$ corresponds to the section of the convolutional filter associated with the k-th channel of the input image. Consequently, the output $\mathcal{L}^{GI} \in \mathbb{R}^{n}$ is analogous to the activation map in convolutional layers.

The final step involves generalizing the output features as well, allowing the layers to produce any arbitrary number $F \geq 1$ of output features for each node. In this case the function $\mathcal{L}_3^{GI} : \mathbb{R}^{n \times K} \to \mathbb{R}^{n \times F}$ is defined as

$$\mathcal{L}_3^{GI}(X) = \boldsymbol{f}\left(\widetilde{\boldsymbol{W}}^{T} \text{vertcat}(X) + B\right),$$

where

- $X \in \mathbb{R}^{n \times K}$ is the input matrix as before;
- $\widetilde{\boldsymbol{W}} \in \mathbb{R}^{nK \times F \times n}$ is a weight tensor defined as the concatenation along the column dimension of the matrices $\widetilde{W}^{(1)}, ..., \widetilde{W}^{(F)}$; these last ones are defined, for each $l = 1, ..., F$, as

$$\widetilde{W}^{(l)} := \begin{bmatrix} \widehat{W}^{(l,1)} \\ \vdots \\ \widehat{W}^{(l,K)} \end{bmatrix} = \begin{bmatrix} diag(\boldsymbol{w}_{\cdot 1}^{(l)})\hat{A} \\ \vdots \\ diag(\boldsymbol{w}_{\cdot K}^{(l)})\hat{A} \end{bmatrix} \in \mathbb{R}^{nK \times n},$$

  where $w_{\cdot k}^{(l)} \in \mathbb{R}^{n}$ is the weights vector associated at the k-th input feature and at the l-th output feature, actually it is the basic filter that describe the contribution of that input feature to the computation of that specific output feature;
- $\widetilde{\boldsymbol{W}}^{T} \text{vertcat}(X)$ is a tensor-vector product;
- $b_{\cdot l}$ is the bias vector connected to the l-th output feature and $B \in \mathbb{R}^{n \times F}$ is the biases matrix with $b_{\cdot l}$ as columns.

For a better understanding of the construction of the tensor $\widetilde{\boldsymbol{W}}$, refer to Figure 2.7.

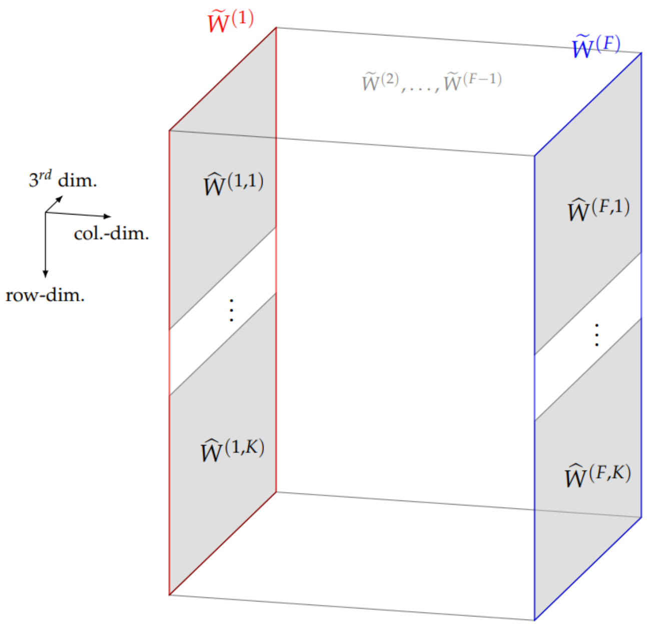


Figure 2.7: Tensor $\widetilde{\boldsymbol{W}}$ where the column are the matrices $\widetilde{W}^{(1)}, \dots, \widetilde{W}^{(F)} \in \mathbb{R}^{nK \times 1 \times n}$ [5].

Given this general formulation, the authors in [5] also present additional properties of the L^{GI}, such as a modified pooling operation and the application of a mask to the reference graph. Furthermore, in the paper by Della Santa [9], a L^{GI} definition is presented for sub-matrices of A, focusing on the connections between two subsets of nodes $(V_1, V_2) \in V$.

To conclude we have to highlight that in a L^{GI}, the total number of parameters is $nKF + nF$, which represents the sum of the number of weights and biases. But, for a L^{FC} with an input size of n and an output size of M, the number of parameters is $nM + M$. Therefore, when $M = n$ and $(KF + F) < (n + 1)$, L^{GI} require fewer parameters to train. This is a significant observation, especially when dealing with very large graphs G (i.e., $n \gg 1$).
It should be noted that the basic implementation of L^{GI} is inefficient due to dense memory allocation. However, the paper [9] presents a sparse implementation of L^{GI} that significantly reduces memory usage.

3 Parameterized Partial Differential Equations

In this chapter, we present the problem of parametrized Partial Differential Equations (PDEs), which is the central issue addressed in this thesis.
In the first section (3.1), we provide an overview of PDEs, explaining what they are, how they are classified, and presenting some of the most well-known and practically applied equations. In the second section (3.2), we introduce parametric PDEs, with formal mathematical formulation, followed by a detailed presentation of the theoretical problem we aim to solve. We conclude the chapter by presenting the solving methods for PDEs, with particular attention to Neural Networks (NNs) (3.3).

3.1 Partial Differential Equations

PDEs are equations that involve rates of change with respect to more than one independent variable. In contrast to Ordinary Differential Equations (ODEs), which deal with functions of a single variable, PDEs describe functions of multiple variables and the relationships between their partial derivatives. They are fundamental in describing various physical, biological, and engineering processes, including heat conduction, wave propagation, fluid flow, and quantum mechanics.

As reported in [31] and [38], a general PDE is expressed as:

$$F\left(x_1, ..., x_n, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1^2}, ..., \frac{\partial^2 u}{\partial x_n^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, ...\right) = 0;$$

where:

- $u = u(x_1, x_2, \ldots, x_n)$ is the unknown function of n variables $x_1, ..., x_n$;
- $\dfrac{\partial u}{\partial x_i}$ represents the first partial derivative of u with respect to x_i;
- $\dfrac{\partial^2 u}{\partial x_i \partial x_j}$ represents the second partial derivative of u with respect to x_i, x_j.

For the ease of notation, and according to literature, from now on we will drop the dependency of u from its variables; i.e., $u := u(x_1, ..., x_n)$.

Classification

PDEs can be classified according to several criteria, as order, linearity and nature of the characteristic equation.
First of all, we can classify a general PDE according to the order of the highest derivative present in the equation. If only first derivatives appear, we call it *first-order PDE*

$$F\left(x_1, ..., x_n, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_n}\right) = 0;$$

if also second derivatives appear, we have a *second-order PDE*

$$F\left(x_1, ..., x_n, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1^2}, ..., \frac{\partial^2 u}{\partial x_n^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, ...\right) = 0;$$

Another classification criterium regards the linearity. For example, a second order PDE ($n = 2$), is *linear* if it can be written as a linear combination of the unknown function and its derivatives:

$$a(x_1, x_2)\frac{\partial^2 u}{\partial x_1^2} + b(x_1, x_2)\frac{\partial^2 u}{\partial x_2^2} + c(x_1, x_2)u = f(x_1, x_2);$$

and it is *nonlinear* if some nonlinear terms are involved, for example

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

The example above is useful to introduce the typical notation adopted in PDEs and, from now on, in this thesis. If the function u represents a time-dependent physical phenomenon, the variable representing the time is denoted by t; all the remaining variables are denoted by $x_1, ..., x_n$. Typically, as in the cases addressed in this thesis, the variables $x_1, ..., x_n$ denotes the spatial variables, i.e., the variables of the physical space of the phenomenon (usually, we have $n \leq 3$).

For second-order linear PDEs, we can further classify according to the nature of the characteristic equation, determined by the discriminant[1] of the quadratic form in the second derivatives. Below we list the three main ones, limited to the case of $n = 2$ for the ease of notation.

- *Elliptic PDE*: the discriminant is negative, such as the Laplace's equation in 2D
$$\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = 0;$$

[1] The discriminant of a quadratic form $ax^2 + bxy + cy^2$ is given by $\Delta = b^2 - 4ac$, and it provides information about the nature of the conic section represented by the quadratic form, such as whether it is an ellipse, parabola, or hyperbola.

- *Parabolic PDE*: the discriminant is zero, such as in the heat equation in 1D

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2};$$

- *Hyperbolic PDE*: the discriminant is positive, such as in the wave equation in 1D

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}.$$

In this thesis, we will focus specifically on stationary and linear problems.

Common Equations

PDEs are frequently used in modeling dynamic systems across various fields, from physics and engineering to finance and biology; in this brief paragraph we present the most well-known and practically applied ones.

Laplace's equation is an elliptic PDE that describes steady-state phenomena, such as electrostatics or incompressible fluid flow. In 2D space, it takes the form:

$$\Delta u = 0,$$

where Δ is the Laplace operator, defined as:

$$\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}.$$

Poisson's Equation is a generalization of Laplace's equation, that includes a source term and describes potential fields affected by external charges or masses. In 2D space, the equation becomes:

$$\Delta u = f(x_1, x_2),$$

where $f(x_1, x_2)$ represents the source term (for example, a charge of density in electrostatics).

Heat Equation is a parabolic PDE that models the distribution of heat (or diffusion of particles) over time in a given region. In one spatial dimension, it takes the form:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2},$$

where α is the thermal diffusivity and $u(x, t)$ is the temperature distribution.

Wave Equation is an hyperbolic PDE that governs the behavior of waves, such as sound waves or electromagnetic waves. In one spatial dimension, the equation is:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{3.1}$$

where c is the speed of wave propagation and $u(x, t)$ is the wave function.

Boundary and Initial Conditions

PDEs usually require additional information to determine a unique solution. This comes in the form of Boundary Conditions (BCs) and Initial Conditions.

In mathematics, BCs can be applied both to ODEs or PDEs. The two most common types are Dirichlet conditions and Neumann conditions.
The *Dirichlet boundary condition* imposes the fixed values that the solution takes along given subsets of the domain's boundary. The problem of finding such solutions is known as the Dirichlet problem, named after Peter Gustav Lejeune Dirichlet (1805–1859) [31]. In the sciences and engineering, this type of boundary condition may also be referred to as boundary condition of the first type or a fixed boundary condition.
The *Neumann boundary condition* specifies the values of the derivatives at other given subsets of the domain's boundary. It is also known as a second-type boundary condition and is named after Carl Neumann (1832-1925) [31].

Example

Let $\Omega \subset \mathbb{R}^n$ be a domain and $\partial\Omega$ its boundary. We denote as Γ_D and Γ_N two subsets of $\partial\Omega$ such that $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$, see Figure 3.1.
Given the PDE,

$$\Delta u + u = 0,$$

a general Dirichlet boundary condition with respect to a function $f : \Omega \to \mathbb{R}$ is

$$u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Gamma_D \subset \partial\Omega.$$

On the other hand, a general Neumann boundary condition with respect to a function $g : \Omega \to \mathbb{R}$ is

$$\frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Gamma_N \subset \partial\Omega,$$

where $\boldsymbol{n} \coloneqq \boldsymbol{n}(\boldsymbol{x})$ typically denotes the normal to $\partial\Omega$ at $\boldsymbol{x} \in \partial\Omega$.

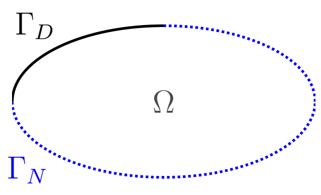


Figure 3.1: *Representation of a general domain Ω with Dirichlet BCs (Γ_D) in a black solid line and Neumann BCs (Γ_N) in a blue dotted line.*

Initial conditions are required for time-dependent PDEs and specify the state of the system at $t = 0$.
For example, given a domain Ω, an initial condition for the wave equation (3.1), might be:

$$u(\boldsymbol{x}, 0) = f(\boldsymbol{x}), \quad \frac{\partial u}{\partial t}(\boldsymbol{x}, 0) = g(\boldsymbol{x});$$

with respect to the functions $f : \Omega \to \mathbb{R}$ and $g : \Omega \to \mathbb{R}$.

3.2 Parametrized Differential Equations

Parametrized PDEs models are widely used across engineering and applied sciences to represent both steady and unsteady phenomena such as heat and mass transfer, solid and fluid mechanics, acoustics, electromagnetics, and financial problems. The characteristic of the parametrized problems is that these phenomena incorporate several input parameters that determine the behavior of the physical phenomenon. For example, these parameters may include geometric configurations, BCs, physical properties, or source terms.
The model used to solve this type of problem must incorporate these input parameters to define the specific physic problem and its corresponding variations, and it must also implicitly link these inputs to key outputs of interest [37].

After this brief introduction to parameterized problems, we present the problem from a mathematical perspective. We first introduce the general formulation, followed by the specific problem we aim to solve in this thesis.

3.2.1 Parametric Weak Formulation

Let us introduce a physical domain $\Omega \subseteq \mathbb{R}^d$, where d is the spatial dimension. Let $\partial\Omega$ be the boundary, we define Γ^D and Γ^N as subsets of $\partial\Omega$, over which we impose the BCs. In particular the Dirichlet BCs on Γ^D and the Neumann BCs on Γ^N.
Let us also introduce the functional Hilbert spaces $\mathbb{V}$,

$$\mathbb{V} = \mathbb{V}(\Omega) = \{v \in H^1(\Omega) | \quad v|_{\Gamma^D} = 0\},$$

where in general $H^1_0(\Omega) \subset \mathbb{V} \subset H^1(\Omega)$ [2] with $H^1_0(\Omega) = \mathbb{V}$ for $\Gamma^D = \partial\Omega$.
$\mathbb{V}$ is characterize by an inner product $(w, v)_{\mathbb{V}}$, $\forall w, v \in \mathbb{V}$ and an induced norm $||w||_{\mathbb{V}} = \sqrt{(w, w)_{\mathbb{V}}}$, $\forall w \in \mathbb{V}$.
Finally, we define the closed parameter domain as $\mathcal{P} \in \mathbb{R}^P$ where $\boldsymbol{\mu} = (\mu_1, ..., \mu_P) \in \mathcal{P}$ is a parameter and $v(\boldsymbol{\mu})$ is a parametric field variable.

Now, we present the general stationary problem in its weak formulation according to the formulation present in [37].
Let $f : \mathbb{V} \times \mathcal{P} \to \mathbb{R}$ be a parametrized linear form with respect to the first variable, and let $a : \mathbb{V} \times \mathbb{V} \times \mathcal{P} \to \mathbb{R}$ be a parametrized bilinear form, where the bilinearity is with respect to the first two variables.
Given a parameter $\boldsymbol{\mu} \in \mathcal{P}$ we are looking for $u(\boldsymbol{\mu}) \in \mathbb{V}$ such that

$$a(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = f(v; \boldsymbol{\mu}), \quad \forall v \in \mathbb{V}.$$

The problem just presented can be greatly simplified, imposing the following two conditions:

- $f(\cdot; \boldsymbol{\mu})$ for all $\boldsymbol{\mu} \in \mathcal{P}$ is a regular $L^2(\Omega)$ source term[3];
- $a(\cdot, \cdot; \boldsymbol{\mu})$ is symmetric for all $\boldsymbol{\mu} \in \mathcal{P}$ (this is not the case of our test cases).

As mentioned before, $\mathbb{V}$ is an Hilbert space and is characterize by an intrinsic norm $|| \cdot ||_{\mathbb{V}}$. Given a fixed parameter $\hat{\boldsymbol{\mu}} \in \mathcal{P}$, the norm induced by the bilinear form a often coincides with the intrinsic norm:

$$(w, v)_{\mathbb{V}} = a(w, v; \hat{\boldsymbol{\mu}}), \quad \forall w, v \in \mathbb{V},$$

$$||w||_{\mathbb{V}} = \sqrt{a(w, w; \hat{\boldsymbol{\mu}})}, \quad \forall w \in \mathbb{V}.$$

[2] The space $H^1(\Omega)$, also known as the Sobolev space H^1 on a domain $\Omega \subset \mathbb{R}^n$, is a function space that consists of functions which, together with their first-order derivatives, are square-integrable [31].

[3] $L^2(\Omega)$ refers to a specific space of functions that are square-integrable over the domain Ω [31].

In order to make these formulations parameter-dependent, we have to make more assumptions in addition to the linearity of f and to the bilinearity of a. So, let $a(\cdot,\cdot;\boldsymbol{\mu})$ be coercive and continuous and $f(\cdot;\boldsymbol{\mu})$ be continuous, both for all $\boldsymbol{\mu} \in \mathcal{P}$ and with respect to the norm $||\cdot||_{\mathbb{V}}$.
For more mathematical details on this section, refer to the book [37].

Discretization

Given the previous parametric weak formulation, this paragraph provides an abstract framework for a discrete approximation, according to the formulation present in [37].
The aim is to seek for an approximate solution in a discrete approximation space $\mathbb{V}_\delta \subset \mathbb{V}$. This space can be constructed with various approaches, for example, we can use piece-wise linear basis functions, standard Finite Elements Methods (FEM), or Spectral Methods.
Given a parameter $\boldsymbol{\mu} \in \mathcal{P}$, the discrete problem involves finding $u_\delta(\boldsymbol{\mu}) \in \mathbb{V}_\delta$ such that

$$a(u_\delta(\boldsymbol{\mu}), v_\delta; \boldsymbol{\mu}) = f(v_\delta; \boldsymbol{\mu}), \quad \forall v_\delta \in \mathbb{V}_\delta.$$

Usually, when the error $||u(\boldsymbol{\mu}) - u_\delta(\boldsymbol{\mu})||_{\mathbb{V}}$ is acceptable, the approximation $u_\delta(\boldsymbol{\mu})$ is considered accurate. But, the practical problem is that the computation of the solution can be very computational expensive. Given $N_\delta = \dim(\mathbb{V}_\delta)$, the discrete space may involve multiple degrees of freedom according to N_δ to obtain the desired accuracy.

A surrogate model like the one proposed in this thesis can help address this issue. Indeed, with many simulations to solve, a NN can handle a large number of them in a significantly shorter time compared to using standard methods.

3.2.2 Formulation of the Problem for the Test Cases

In this paragraph, we provide a detailed presentation of the theoretical problem we aim to solve, describing it in terms of a continuous problem. The formulation we present is used to describe a parametric PDE where also the BCs are parameterized.
It is important to note that, even if we present the problem in its continuous form, for each test case, simulations were performed in a discrete environment, and the ground truth values were determined using the traditional FEM.

Before we begin, we must emphasize that the notation we use here is the one introduced in Section 3.2.1, so we have the functional Hilbert spaces $\mathbb{V}$ and the parameter domain $\mathcal{P}$.

The domain we work with, Figure 3.1, is an open, bounded, and regular spatial domain, which we denote by $\Omega \subset \mathbb{R}^2$. Given its boundary $\partial\Omega$, we define two subsets, Γ_D and Γ_N, where Dirichlet and Neumann BCs are imposed, respectively. We also assume that $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

The mathematical problem we want to solve is the one presented in Section 3.2.1, with the linear form f and the bilinear form a. The parameter we consider is $\boldsymbol{\mu} \in \mathcal{P}$ and it has the following generic form $\boldsymbol{\mu} = [\mu_1, .., \mu_k, \mu_D, \mu_N]$, where

- $\mu_1, .., \mu_k$ represent the physical parameters of the problem (in some cases, they can also be the values of the BCs);
- μ_D indicates where the Dirichlet BCs are applied, defining the boundary subset $\Gamma_D^{\mu_D}$;
- μ_N indicates where the Neumann BCs are applied, defining the boundary subset $\Gamma_N^{\mu_N}$.

Moreover, we define the value of the Dirichlet and Neumann BCs with the variables z_D and z_N, respectively. In this setting, we denote the duality pairing between $\mathbb{V}$ and $\mathbb{R}$ with $\langle \cdot, \cdot \rangle$ and so we define the forcing term as

$$\langle F(\boldsymbol{\mu}), v \rangle = \langle f(\boldsymbol{\mu}), v \rangle + \langle f_N(\boldsymbol{\mu}), v \rangle + \langle f_D(\boldsymbol{\mu}), v \rangle ,$$

where

$$\langle f_N(\boldsymbol{\mu}), v \rangle = \int_{\Gamma_N^{\mu_N}} z_N v \text{ ds} \quad \text{and} \quad \langle f_D(\boldsymbol{\mu}), v \rangle = \int_{\Gamma_D^{\mu_D}} z_D v \text{ ds}.$$

As example, Figure 3.2 shows a generic spatial domain Ω under the action of two parameters $\mu^1 = [\mu^1, \mu_N^1, \mu_D^1]$ and $\mu^2 = [\mu^2, \mu_N^2, \mu_D^2]$. The parameters define two different regions, $\Gamma_N^{\mu_N}$ and $\Gamma_D^{\mu_D}$, representing Neumann and Dirichlet BCs.

3.3 Solving methods for PDEs and NNs

There are several methods for solving PDEs, both analytically and numerically. Among the traditional and analytically ones, we can mention Separation of Variables and Fourier transformation. The Separation of Variables method assumes the solution can be written as the product of functions, each dependent on a single variable, $u(x, t) = X(x)T(t)$. Substituting into the PDE and separating the variables often leads to simpler Ordinary Differential Equations (ODEs) to solve [17]. It is also possible to use the Fourier series and transformations. These methods decompose a function into sinusoidal components, which are easier to handle analytically [17].

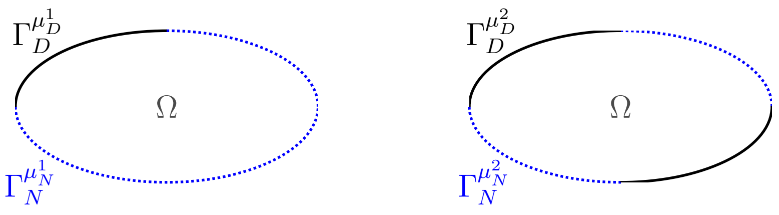


Figure 3.2: Representation of a general domain Ω for two values of μ_N and μ_D, i.e., μ_N^1 and μ_D^1 (left) and μ_N^2 and μ_D^2 (right). The boundary where Neumann is applied is denoted by a blue dotted line, while boundary where Dirichlet is applied is denoted by a black solid line.

Unfortunately, these analytical methods have limited applicability because in most cases, obtaining a solution in an explicit closed form is not feasible. This is why numerical methods are more commonly used than analytical ones. The underlying concept of numerical approaches is to determine substitute functions that approximate the original unknown functions, with two properties: first, maintaining an error acceptable for practical purposes, and second, being relatively easy to compute [31]. Given u the exact solution of the PDEs a general numerical method try to construct an approximation of u, u_N, in such a way the error $u_N - u$ is acceptable. In this setting $N \geq 1$ represents the finite dimension of the numerical problem.
As discussed in the book [31], numerical methods can be grouped into several categories, which are outlined below.

- *Finite Difference Method (FDM)*:
  It discretize the domain into a grid and approximate the derivatives in the governing equations with finite differences. It is often used for problems in one-dimensional or simple geometries because it requires structured grids and is easier to implement for regular domains.

- *Finite Element Method (FEM)*:
  It divide the domain into smaller, simpler parts called "elements" (usually triangles or quadrilaterals in 2D, tetrahedra or hexahedra in 3D) and the solution is approximated by a piecewise polynomial function over these elements. It is highly flexible and suitable for complex geometries and unstructured meshes. This is the method we used to generate the data for our test cases, see Section 4.2.

- *Finite Volume Method (FVM)*:

It divide the domain into small control volumes (cells), and the fluxes across the cell boundaries are computed to ensure the conservation of quantities such as mass, momentum, or energy. It is widely used in computational fluid dynamics due to its ability to handle conservation laws and adapt well to irregular meshes.

- *Spectral Method (SM)*:
  Unlike previous methods, such as FEM or FDM, spectral method use global approximation, meaning the basis functions are nonzero over the entire domain, and the solution is approximated as a sum of global basis functions (usually trigonometric functions like sines and cosines). It is commonly used in problems with periodic BCs or problems where the solution is smooth over the domain. It struggle with complex geometries or solutions with sharp discontinuities.

- *Spectral Element Method (SEM)*: It is a higher-order version of the FEM, combining the geometrical flexibility of FEMs with the high accuracy of SMs. The solution is approximated using high-degree polynomials within each element, providing greater accuracy with fewer elements compared to FEM. It is often used for solving problems in fluid dynamics, electromagnetism, and other fields requiring high precision.

For more information on this numerical methods, the reader can refer to the seminal book [31].

Traditional methods for solving PDEs, such as the ones just presented, often require a fine discretization of the spatial domain. This leads to a computationally expensive process, especially when the dimension of the parameter space is large or when the PDE domain is characterized by a high geometrical complexity; therefore, finding solutions in real-time or over large parameter ranges is a challenging task. Furthermore, in many cases, obtaining accurate solutions to PDEs requires solving the problem iteratively, which becomes impractical due to the enormous computational costs involved.
Thus, although numerical methods work fairly well, they still present critical issues, including the high computational cost and the fact that the analysis must be repeated from scratch every time even a single parameter is changed.

3.3.1 Numerical Methods for Parametric PDEs

The numerical solution for parameterized PDEs using conventional high-fidelity methods, such as the ones mentioned in this section's introduction (FEM, FVM, SEM), is often impractical in scenarios involving many queries or real-time computations [28]. Indeed in these cases, it is essential to efficiently and accurately compute an output of interest while varying the input parameters. However, the complexity and high computational expense of solving the full partial differential equation for each new parameter make a direct approach impractical.As a result, some alternative methods have been explored that allow obtaining the desired output with minimal computational cost, without compromising the predictive accuracy of the detailed model [37].

Reduced Order Models (ROMs) [3] offer an alternative approach that can significantly lower the computational demands (both storage and CPU time) required for analyzing and simulating these equations. Over the past decades, ROMs have evolved into a widely recognized set of techniques built on strong mathematical foundations.
One of these methods, the Reduced Basis method [37], facilitates fast and accurate evaluations of the solution for different parameter values. To construct the reduced space that enables these efficient computations, techniques like Proper Orthogonal Decomposition (POD) or the Greedy algorithm are commonly used [28]. The POD is an SVD-based approach for identifying the principal components, while the Greedy algorithm iteratively expands the space by adding basis functions corresponding to the poorest approximation in the parameter space [31]. These methods separate the computations into two phases, online and offline, ensuring a predictable and quite good accuracy during the online phase with minimal computational cost. However, these approaches are typically linear, and their efficiency tends to decrease when dealing with models that are difficult to reduce or contain non-affine or non-linear terms [28].

Some of the limitations of the traditional approaches can be address by investigating non-intrusive, fast and efficient model order reduction techniques through a DL lens [21]. Leveraging nonlinear machine learning approaches facilitates the discovery of a low-dimensional representation of the latent subspace and captures feature correlations, thanks to its superior ability to learn underlying patterns.

3.3.2 Neural Networks for PDEs

In recent years, deep NNs, known for their exceptional ability to handle and predict complex systems, have been extensively applied across various domains, and in engineering numerical simulations, NNs have been applied as direct solvers for approximated systems, offering highly efficient solutions [18]. Due to their computational efficiency and scalability, particularly on heterogeneous platforms, NNs have emerged as a promising tool in scientific computing, even enabling real-time PDE solving [20].

For example, in the article [28], we can find a parallelism between one of the key components of NNs, the autoencoder architecture, and the ROMs, introduced in the previous paragraph. The paper shows that autoencoder results particularly well-suited for use in ROMs, extending linear compression techniques, like POD. The basic structure of an autoencoder consists of a nonlinear encoding and a decoding, connected via a bottleneck. Comparing the autoencorder with ROMs, the authors have seen this bottleneck as the latent dimension, which functions as the reduced space, where the encoder compresses the information from the high-order system. The decoder then "projects" this reduced representation back to the original dimension [27].

Now, let us take a general look at the history of NNs used in this field.
While the first studies utilized FCNNs, more recent research has progressively adopted optimized architectures that leverage spatial and temporal correlations for more efficient training processes [27].
One architecture that has been extensively examined is CNNs, which efficient detect and learn patterns from the spatial characteristics of the provided data. Initially designed for image classification, CNNs have recently found widespread applications in dynamic modeling and also parameterized PDEs [21], [24] demonstrating quite strong performance. However, CNNs typically operate on structured datasets that resemble images composed of pixels. While this structure is common in computer vision, it is less frequently encountered in the context of physical problems (like those we address in this thesis).
To simulate dynamic behaviors one might use simply Cartesian meshes [26], but irregular meshes are often necessary, as for the PDEs. Indeed, in these cases the control equations are frequently defined on parameterized and complex domains that require unstructured meshes [28]. A potential solution is to transform these grids into image-like representations [12], but there is often inconsistency in the interpretation. In fact, it is challenging to establish an efficient ordering and reshaping that aligns with the physical aspects of the problem [28]. Alternatively, interpolation and level set approximations can be used to convert the data into structured meshes. For instance, in [36], the authors suggest a preprocessing step using k-PCA to obtain a 2D representation of the data. However, this method necessitates dense meshes to retain information during the reshaping process, thus increasing computational costs.
Algorithms have gradually evolved to adapt to the geometry of the domain, as the mesh-informed NNs [11] and the continuous convolutional filters [8], [10]. These works advocate for a rethinking of NN architectures to incorporate geometric information while maintaining a physically consistent inductive bias. Finally, it has been observed that GNNs provide a natural framework for analyzing PDE solutions defined on unstructured meshes [28].
From the literature, we can present the reader with some works in this field. We

can start by mentioning Ray et al. [33] whose proposed a NN-based indicator to correct irregular solutions within the discontinuous Galerkin scheme, and Chan et al. [7] or Wang et al. [41] whose utilized NNs to address multiscale problems.

In terms of network architecture, traditional NN models suffer from network degradation, making it difficult to achieve high accuracy. Increasing the network's depth does not necessarily improve accuracy and can even have a detrimental effect [16]. To address this issue, the residual architecture has been introduced; it includes a short connection, called residual connection, that spans the hidden layers, ensuring the transfer of training information to each layer (for more information see Section 2.1). In numerical computation of PDEs, Tong et al. [30] applied ResNet to simulate linear and nonlinear self-consistent systems. Also Jiang et al.[16] utilize a highly efficient ResNet architecture to address the solution of PDEs. Firstly, they incorporated the ResNet architecture into the NN model, allowing accuracy to improve as the network depth increases, and then the NN model was integrated with a correction iteration process, which iteratively reduces the error in the NN's results.

The idea of using NNs to obtain numerical solutions for PDEs also introduces certain risks. According to the universal approximation theorem [40], a piecewise continuous objective function is a necessary condition for reliable NN approximations. However, solutions to PDEs, particularly nonlinear ones, often do not meet this regularity condition. As a result, direct NN predictions frequently lack sufficient numerical accuracy [16].
To address this, many NN-based approaches focus on semi-analytic models rather than direct prediction. For instance, Raissi et al.[32] introduced the physics-informed neural network (PINN) to solve PDEs by incorporating integral forms into the loss function. Ehsan et al. [19] developed the hp-VPINN, an enhanced version of PINN with improved accuracy. These methods translate the original PDEs into parametric models and design penalty functions to ensure compatibility with the original PDEs, thereby improving the accuracy of NN-based solutions. Using NNs to solve intermediate linear equations is another promising approach that combines the strengths of NNs with traditional numerical methods. For example, Xiao et al.[44] introduced a NN-based solver to accelerate the FDM process for solving the Poisson equation in fluid simulations. In this context, the network architecture and the regularity of the linear equations play a crucial role, with the latter typically being more manageable and easier to quantify compared to the original PDEs or their parametric models. As a result, applying NNs to solve intermediate linear equations has the potential to offer greater precision improvements [16].

Interpretability is another significant challenge in applying NNs to PDEs. It refers to understanding how a specific input leads to a particular output in a NN and what information the network learns during training. EXplainable Artificial Intelligence (XAI) is a new frontier that addresses this problem by striving to make AI systems transparent, interpretable, and more trustworthy, but the NNs interpretability remains an issue, which limits their use in numerical computations [4]. However, NNs have been shown to be reliable as internal interpolation when the train set is sampled uniformly enough to cover almost all possible inputs. Fortunately, homogeneous datasets for linear equations are easy to generate [16]. Additionally, for a given algorithm, the discretization scheme of the PDE is typically unique, restricting the resulting linear equations to a narrow range. This helps NN-based solvers for intermediate linear equations bypass the interpretability challenge effectively [16].

We now focus on the approach taken in this thesis. As previously mentioned the curse of dimensionality is a common challenge in numerical PDEs, particularly when uncertainties are represented as random coefficients within the equations. However, often the variability of physical quantities derived from a PDE can be effectively captured using just a few features in the space of coefficient fields. Based on this observation, the thesis will employ the GINN model discussed in the previous Section 2.3, applying it to a parametric PDE problem where the physical quantities of interest are treated as input coefficients. The idea is to train the model on many solutions obtained from given parameter values, in order to permit a fast evaluation (i.e., prediction) of the solution given new parameter values.

4 Test Cases and Methodology

This chapter starts by presenting the actual physical problems that we aim to solve and then outlines the methodology adopted in this work, providing the reader with more information regarding the practical aspects of the research. After presenting the two test cases, we briefly describe how the two datasets have been created and used. Subsequently, we proceed by explaining how we designed the NNs architectures and selected the hyperparameters. Towards the end of the chapter, we describe the evaluation methods used for the analysis, and in the final part, we provide details regarding the actual implementation.

4.1 Test Cases

The general mathematical formulation describing the problems addressed in this thesis was presented in Section 3.2.2. In this section, we specifically introduce the two test cases. As mentioned in the introduction, the physical problems we examined are two, one is purely diffusion-based, and the other involves both diffusion and convection.
These two physical phenomenons are fundamental processes that describe the transport of substances and energy in various physical systems, from environmental processes to industrial applications.
Diffusion refers to the gradual spreading of particles from regions of high concentration to regions of low concentration. A common example is an oil spill on water, where the oil slowly spreads over the surface. Another classic case is heat diffusion, where thermal energy spreads from hotter areas to cooler ones, leading to temperature equalization over time.
Convection, on the other hand, involves the transport of substances or heat through the movement of a fluid. A typical example is the effect of wind blowing across a landscape. The direction and intensity of the wind determine how air and heat are distributed. In convection-dominated systems, the movement of the fluid plays a more significant role than diffusion in determining how substances or heat are transported.

To make the text clearer, from now on, we will refer to the first problem as *Test 1* and the second as *Test 2*.

Test 1

The physical domain of this case test is the unit square with a circular hole in the center, i.e., $\Omega \doteq \{(0,1) \times (0,1)\} \setminus \{(x_1, x_2)\} \in \mathbb{R}^2$ such that $(x_1 - 0.5)^2 + (x_2 - 0.5)^2 - 0.3^2 < 0\}$. The physical phenomenon being discussed is diffusion, described by the following simple equation, $-\Delta u = 0$ in Ω. In this case, heat diffuses from the circular hole in the center, and its emission is selective, depending on the Neumann BCs imposed on the edge of the inner circle.

Given $\partial\Omega$ the domain boundary, we can identify two portions of it, the inner circular boundary, $\partial\Omega_c$, and the external square boundary, $\partial\Omega_s$, such that $\partial\Omega_c \cup \partial\Omega_s = \partial\Omega$ and $\partial\Omega_c \cap \partial\Omega_s = \emptyset$.
In this setting, we apply four different kinds of BCs, as depicted in Figure 4.1:

- $\Gamma_N^{\mu_N} \subseteq \partial\Omega_c$, non-homogeneous Neumann BCs on part of the inner circular boundary, according to parameter μ_N;
- $\overline{\Gamma}_N^{\mu_N} \subseteq \partial\Omega_c$, homogeneous Neumann BCs on part of the inner circular boundary, according to parameter μ_N;
- $\Gamma_N \subset \partial\Omega_s$, homogeneous and fixed Neumann BCs on the right and left edges of the square boundary;
- $\Gamma_D \subset \partial\Omega_s$, homogeneous and fixed Dirichlet BCs on the top and bottom edges of the square boundary;

where $\partial\Omega_c = \Gamma_N^{\mu_N} \cup \overline{\Gamma}_N^{\mu_N}$, $\Gamma_N^{\mu_N} \cap \overline{\Gamma}_N^{\mu_N} = \emptyset$, $\partial\Omega_s = \Gamma_D \cup \Gamma_N$ and $\Gamma_D \cap \Gamma_N = \emptyset$.
The involved parameter is $\boldsymbol{\mu} = [\mu_c, \mu_N]$, where μ_N maps each node of the mesh on the inner circle $\partial\Omega_c$, to the subsets $\Gamma_N^{\mu_N}$ or $\overline{\Gamma}_N^{\mu_N}$. If the node is in $\Gamma_N^{\mu_N}$ we apply an homogeneous Neumann condition equal to 0, while if it is in $\overline{\Gamma}_N^{\mu_N}$ we apply a non-homogeneous Neumann condition equal to $\mu_c \in (0,1)$. Hence, these parameters model how heat is emitted from the inner circle to the rest of the domain.

Given a generic parametric instance $\boldsymbol{\mu}^* = [\mu_c^*, \mu_N^*]$, the problem we want to solve appears as follows:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma_D, \\ \dfrac{\partial u}{\partial \mathbf{n}} = \mu_c^* & \text{on } \Gamma_N^{\mu_N^*}, \\ \dfrac{\partial u}{\partial \mathbf{n}} = 0 & \text{on } \Gamma_N \cup \overline{\Gamma}_N^{\mu_N^*}, \end{cases} \tag{T1}$$

where $\mathbf{n}$ is the normal outward vector to $\partial\Omega$.

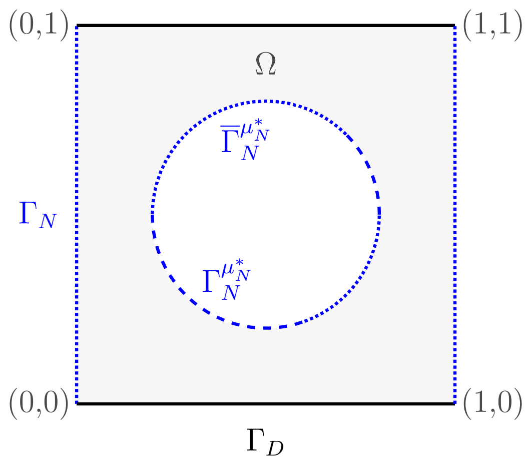


Figure 4.1: Spatial domain Ω: schematic representation for μ_N^*. The homogeneous Dirichlet boundary is denoted by a black solid line (Γ_D); the homogeneous Neumann boundary is denoted by a blue dotted line (Γ_N and $\overline{\Gamma}_N^{\mu_N^*}$) and the non-homogeneous Neumann boundary is denoted by a blue dashed line ($\Gamma_N^{\mu_N^*}$).

Test 2

The physical domain of this case test is simply the unit square $\Omega \doteq \{(0,1) \times (0,1)\}$, and the physical phenomenon being discussed is a problem of diffusion and convection. The equation that describes it, is the following

$$\alpha(\mu_1) \cdot \nabla u - \beta(\mu_2) \cdot \Delta u = 1 \text{ in } \Omega,$$

where $\alpha(\mu_1)\nabla u$ is the convection term, while $\beta(\mu_2)\Delta u$ is the diffusion term.
The involved parameter is $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_D]$, where $\mu_1 \in (0, \pi)$ controls the convection, $\alpha(\mu_1) = (\cos(\mu_1), \sin(\mu_1))$, and $\mu_2 \in$ (0,5) controls the diffusion, $\beta(\mu_2) = 10^{-\mu_2}$. In particular, we use μ_1 to parameterize the direction of convection, as μ_1 represents the angle of the convection vector, i.e., the angle in which "the wind blows". On the other hand, we use μ_2 to parameterize different diffusion or convection regimes. Specifically, if $\mu_2 \to 0$, the case will be diffusion-dominated, whereas if $\mu_2 \to 5$, the case will be convection-dominated. Moreover, the behavior of the physical phenomenon will generally be influenced by the imposed BCs, which determine the characteristics of the domain's boundary.

In this setting, we apply three different kinds of BCs, as depicted in Figure 4.2:

- $\Gamma_D \subset \partial\Omega$, homogeneous and fixed Dirichlet BCs on a specific portion of the perimeter of the square domain $\overline{OA} \cup \overline{OB}$, where $O =$ (0,0), $A =$ (1,0), and $B = (0, 0.25)$;

- $\Gamma_D^{\mu_D} \subset \partial\Omega$, homogeneous Dirichlet BCs on part of the perimeter of the square domain, according to parameter μ_D;

- $\Gamma_N^{\mu_D} \subset \partial\Omega$, homogeneous Neumann BCs on part of the perimeter of the square domain, according to parameter μ_D;

where $\partial\Omega = \Gamma_D \cup \Gamma_D^{\mu_D} \cup \Gamma_N^{\mu_D}$, $\Gamma_D \cap \Gamma_D^{\mu_D} = \emptyset$, $\Gamma_D \cap \Gamma_N^{\mu_D} = \emptyset$ and $\Gamma_D^{\mu_D} \cap \Gamma_N^{\mu_D} = \emptyset$.
The BCs imposed on the boundary are both Neumann and Dirichlet, creating "holes" along the perimeter of the square, which result in different behaviors of the diffusion-convection phenomenon.

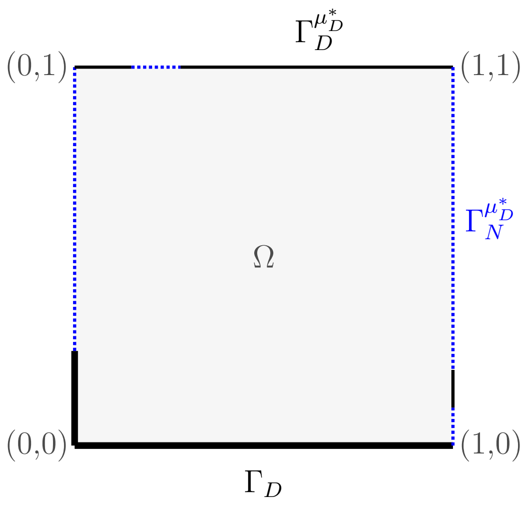


Figure 4.2: Spatial domain Ω: schematic representation for μ_D^*. The homogeneous Dirichlet boundary is denoted by a black solid line, the thicker one is where the conditions are fixed ($\Gamma_D^{\mu_D^*}$ and Γ_D); the homogeneous Neumann boundary is denoted by a blue dotted line ($\Gamma_N^{\mu_D^*}$).

Given the general parameter, $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_D]$, the BCs are determined by μ_D, which maps each node of the boundary to 0 if homogeneous Dirichlet conditions apply and to 1 if the problem features homogeneous Neumann conditions.
The choice to fix a portion of the BCs (Γ_D) was made to guarantee the numerical stability of this problem in the convection-dominated case (i.e., for large values of μ_2), as these models have been simulated using piece-wise linear FEM simulations.

Given a generic parametric instance $\boldsymbol{\mu}^* = [\mu_1^*, \mu_2^*, \mu_D^*]$, the problem we want to solve appears as follows:

$$\begin{cases} \alpha(\mu_1^*)\nabla u - \beta(\mu_2^*)\Delta u = 1 & \text{in } \Omega, \\ \dfrac{\partial u}{\partial \mathbf{n}} = 0 & \text{on } \Gamma_N^{\mu_D^*}, \\ u = 0 & \text{on } \Gamma_D \cup \Gamma_D^{\mu_D^*}, \end{cases} \tag{T2}$$

where $\mathbf{n}$ is the normal outward vector to $\partial\Omega$.

4.2 Data Preparation

For each of our two problems, diffusion and convection-diffusion, we have a system of PDEs, (T1) and (T2), for which, as mentioned in the introduction, it is difficult to find analytical solutions. Among the numerical solution techniques for PDEs, presented in Section 3.3, we used the FEM. With this method, we create a mesh consisting of these "elements", and the solution is interpolated over each element by linear functions, with nodes at the vertices of the elements. For our problems, we used $\mathbb{P}_1$ finite element data, triangles in 2D, and since the $\mathbb{P}_1$ elements are uniquely defined by the nodal values on the triangle [31], the dimension of the space equals the number of nodal values, so the network inputs correspond to the FEM nodal values. In the end, we created a mesh of 1141 points for the first problem and a mesh of 3967 points for the second one.

Now, we briefly describe how we obtained the parameters that characterize our test cases, considering both the physical and geometric ones (BCs). For the first problem, *Test 1*, we have the parameter $\boldsymbol{\mu} = [\mu_c, \mu_N]$, where μ_c is the physical one and μ_N the geometrical one. For the second problem, *Test 2*, we have $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_D]$ where μ_1 and μ_2 are the physical ones and μ_D is the geometrical one. The physical parameters of the problems are derived from simple distributions. For the first problem, $\mu_c \in (0,1)$ is obtained through a uniform distribution [2] between 0 and 1, $\mu_c \sim \mathcal{U}(0, ...,1)$. For the second problem, $\mu_1 \in (0,5)$ is given by a beta distribution [2], $\mu_1 \sim \mathcal{B}(\alpha, \beta)$ with $\alpha = \beta = 0.5$, while $\mu_2 \in (0, \pi)$ is obtained from a uniform distribution between 0 and π, $\mu_2 \sim \mathcal{U}(0, ..., \pi)$.
The geometric parameters, on the other hand, are defined in a somewhat more complex manner, we present the base reasoning. In the first problem, we have the BCs fixed on the square's boundary but not on the internal circumference, where we have a total of $M = 88$ nodes. First, we define some interval centers N_c, where the minimum number of intervals is 1 and the maximum is 5, $N_c \sim \mathcal{U}(1, ...,5)$. These centers c_i are uniformly sampled on the circumference for all N_c, meaning that we randomly select a node uniformly between 1 and M, $c_i \sim \mathcal{U}(c_1, ..., c_M)$. Given L_c, the length of the circle, the length of the interval, or arc of the circumference, is discretely sampled from one node to all nodes up to

the maximum number of intervals considered, so it is uniformly sampled between $[L_c/M, L_c/5]$, $l_i \sim \mathcal{U}(1, ..., L_c/5)$. In the second problem, the reasoning is the same, but every time the region is exited from the $\Gamma_D^{\mu_D^*}$, the interval is cut off. For better understanding, see Figure 4.3.

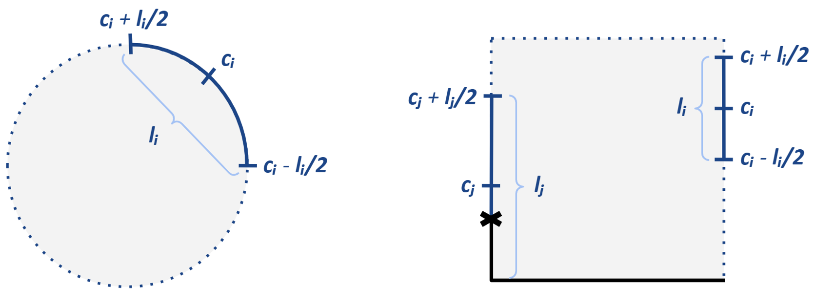


Figure 4.3: Scheme for the creation of the geometric parameters characterizing the BCs for the circular boundary of the first problems and the square boundary of the second problem.

For both problems, 5000 simulations were created, but for the second test case, much more complex than the first one, some simulations were discarded due to inconsistencies. In conclusion, we used a dataset of 5000 simulations for the first problem and a dataset of 4715 for the second problem.

As suggest in literature, both datasets are split into three distinct sets: a training set, a validation set, and a test set. Indeed, splitting data into training, validation, and test sets helps assess the performance a DL model by training it on the first one (training set), monitoring and regularizing the training behaviour on the second one (validation set), and ultimately evaluating its generalization on the independent test set. This separation ensures robustness, prevents overfitting and enhances the model's ability to make accurate predictions on new, unseen data.
For our experiments, we designated a fixed portion of each dataset as the test set, while the remaining data was split into training and validation sets. To evaluate the robustness of the architectures, we trained the models on training sets of varying sizes. For the first problem, *Test 1*, we have a dataset of 5000 simulations, where 3000 are allocated as test set. The training and validation sets are initially sized at 1024 and 512, respectively, then reduced to 512 for training and 256 for validation, and finally to 256 for training and 128 for validation. For the second problem, *Test 2*, we have a dataset of 4715 simulations, where 1500 are assigned as test set, and in this case, we used training sets of 1024 and 512, and corresponding validation sets of 512 and 256.

4.3 Architecture Design

The architectures used in our experiments are essentially two: an FCNN, with and without residual blocks, and a residual GINN. What we request as output is the same in both cases: the value of the solution to the problem for each node of the mesh.
Among the hyperparameters, we have some fixed in both architectures; the specific values are listed in Table 4.1.

Hyperparameter	**Value**
batch size	16
learning rate factor	0.5
learning rate patience	100 epochs
early stopping	750 epochs
optimizer	ADAM with learning rate of 10^{-3}
maximum number of epochs	10000 epochs

Table 4.1: Fixed hyperparameters for both architectures.

In the case of the FCNN, we employ a fairly standard architecture and the hyperparameters we focus on are the number of hidden layers, the width of these layers, the use of batch normalization, and the type of activation function. An additional option for this network is whether to include residual blocks as introduced in Section 2.1. For a better understanding see Figure 4.4.

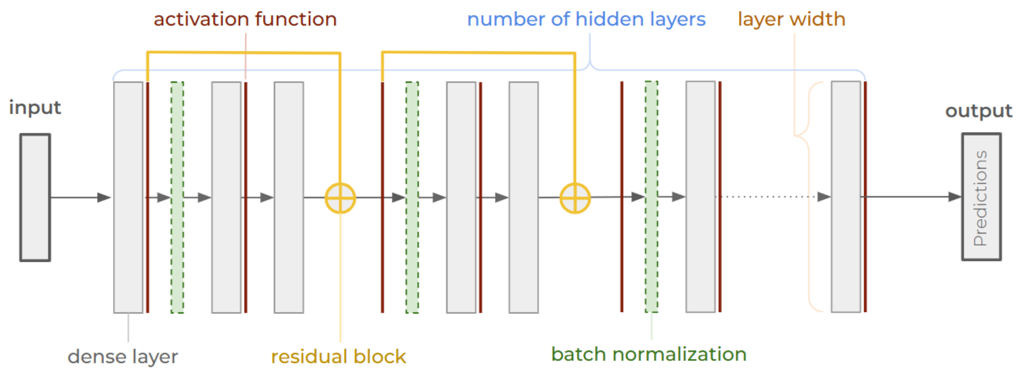


Figure 4.4: Schematized architecture of the FCNN used in our experiments.

Regarding the architecture of the GINN, see Figure 4.5, the base structure is similar to the one just presented but, we use the GI layers, introduced in Section 2.3.1, instead of the dense fully-connected layers.

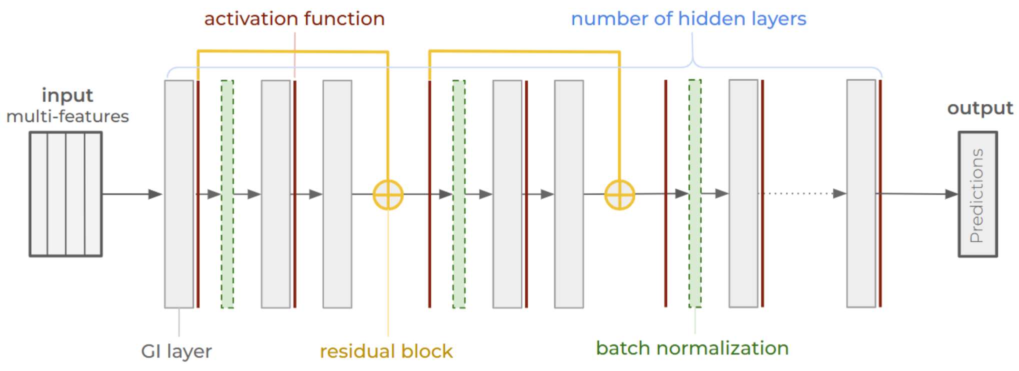


Figure 4.5: Schematized architecture of the GINN used in our experiments.

In addition to the number of hidden layers, the activation function, and the use of batch normalization, we test the network by varying the number of filters, i.e., the number of output features associated with each node of the graph. Additionally, we try with feeding the input data (i.e., all the BCs and physical parameters) again into the network after n layers to see if this improves the network's performance; this technique will be referred to as "input refresh", see Figure 4.6.

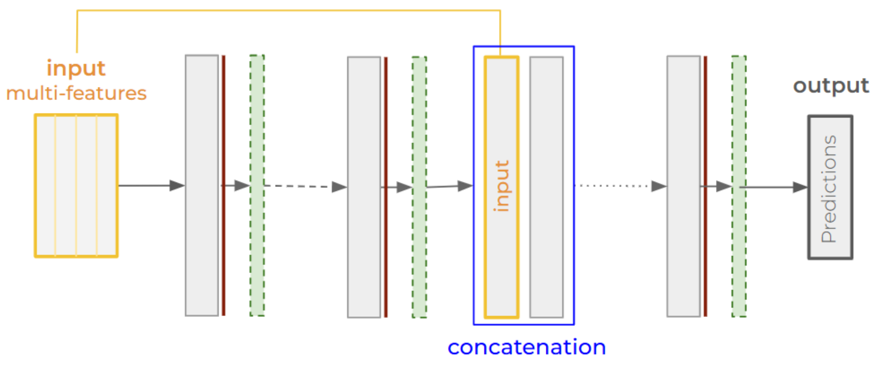


Figure 4.6: Input Refresh mechanism for the GINN architecture.

One of the most important property of this architecture is that it has a multi-features input $\in \mathbb{R}^{N \times K}$, where N is the number of the graph nodes and K the number of features associated to each node. The features we pass to the DL model are the parameters of the problem, the ones regarding the BCs and the ones regarding the physical parameters; for more details see Figure 4.7.
Another key difference between the two architectures is that the number of units in the layers (width of the hidden layer) is a hyperparameter that we choose only in the case of FCNNs, whereas in the case of GINNs, it is always fixed and equal

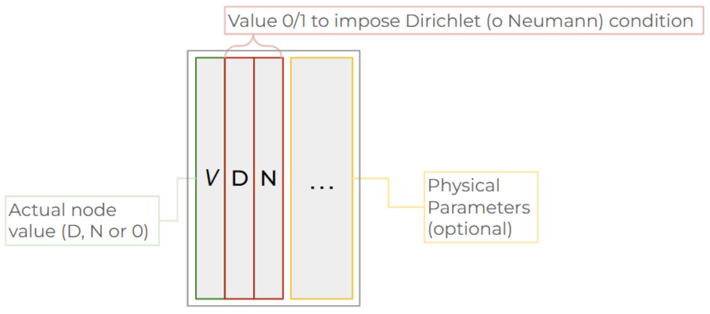

Value 0/1 to impose Dirichlet (o Neumann) condition
V
D
N
...
Actual node value (D, N or 0)
Physical Parameters (optional)

Figure 4.7: Input multi-features of the GINN architecture.

to the number of nodes in the mesh, as presented in Section 4.2.

4.4 Simulations Setup

As discussed in Chapter 3, one of the issues with standard GCNs is that, in many cases, they fail to outperform even a simple MLP. One of the primary goals of the new GINN architectures is to demonstrate that they can indeed surpass the performance of traditional MLPs. To test this, experiments are conducted using both types of NNs on both the two test cases.

For both problems, we perform an extensive grid search for hyperparameters, testing a pair of values for each hyperparameter. Specifically, we explore different values for the number of hidden layers (nh), the type of activation function (af) and the use of batch normalization (bn). For the FCNNs we tested also, the width of the hidden layers (wh) and the use of residual connections (rs), while for the GINNs, the number of filters (nf) and whether or not to refresh the input (rf). The Table 4.2 presents the various combinations tested for both problems.

Hyperparameter	**Test 1**		**Test 2**	
	FCNN	*GINN*	*FCNN*	*GINN*
nh	[6, 8]	[60, 80]	[6, 8]	[80, 100]
bn	[True, False]	[True, False]	[True, False]	[True, False]
wh	[1024, 2048]	-	[2048, 4096]	-
nf	-	[3, 5]	-	[5, 10]
af	[mish, elu]	[mish, elu]	[mish, elu]	[mish, elu]
rs	[True, False]	-	[True, False]	-
rf	-	[15, None]	-	[15, None]

Table 4.2: Hyperparameters grid search.

In conclusion, we have trained, for each problem, 16 basic FC architectures, 16 FC architectures with residual modules, and 32 residual GINNs. Each architecture was trained across the various training and validation sets mentioned earlier. See Figure 4.8 for an overview of all the experiments conducted.

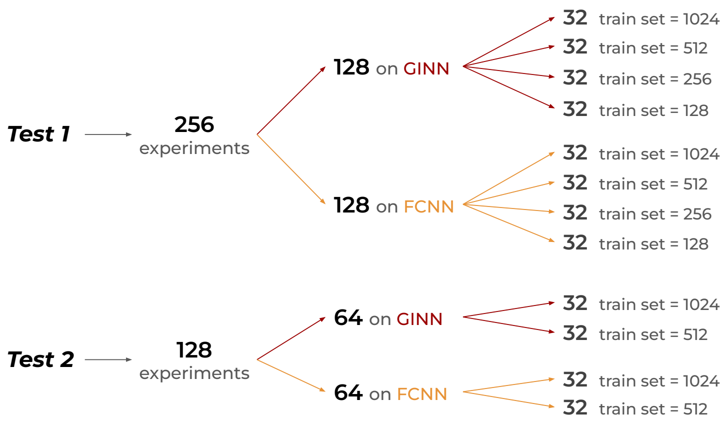


Figure 4.8: Experiments conducted according to the architecture used and the number of ground truth simulations used for training.

4.5 Evaluation

When tackling a regression problem, several metrics can be used to assess the accuracy and quality of the model, each providing different insights into its performance. Let us assume that $n \in \mathbb{N}$ is the number of mesh points and define $\boldsymbol{y} = (y_1, ..., y_n) \in \mathbb{R}^n$ as the values of actual solution and $\hat{\boldsymbol{y}} = (\hat{y}_1, ..., \hat{y}_n) \in \mathbb{R}^n$ as the values of predicted solution. Let us also define the function u interpolated at the nodes of the mesh of the actual solution and the function $\hat{u}$ interpolated at the nodes of the mesh of the predicted solution
It is common to evaluate the results by comparing the actual value y_i with the predicted value $\hat{y}_i$. Nonetheless, since our task is the regression of PDE solutions, we take into account more specific performance measures; e.g., errors based on the L^2 norm and H^1 semi-norm of the functions defined on the domain of the problem, u and $\hat{u}$.
Below is a brief overview of some key metrics that we used in our analysis.

1. **Mean Absolute Error** (MAE):
   MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the test sample

of the absolute differences between prediction and actual observation where all individual differences have equal weight. MAE is intuitive and easy to interpret, providing a clear idea of the average error in the same units as the target variable. The formula is:

$$MAE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|.$$

2. **Mean Relative Range Error** (*MRRE*):
This metric assesses the error in relation to the range of the actual values. It is particularly useful for evaluating models where the range of values is large and can help understand how errors scale with the data's variability. The formula is:
$$MRRE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{y_{\max} - y_{\min}},$$
where $y_{\max}$ and $y_{\min}$ are the maximum and minimum values of the actual solution, respectively.

3. **L^2-norm Error** (*L2E*):
L2E is the error related to the Hilbert space $L^2(\Omega)$ and it is defined by the norm in $L^2(\Omega)$ of the difference between the solutions as functions. The formula is:
$$L2E(u, \hat{u}) = \|u - \hat{u}\|_{L^2(\Omega)},$$
where
$$\|u - \hat{u}\|_{L^2(\Omega)} = \sqrt{\int_{\Omega} (u - \hat{u})^2 \, d\Omega}.$$

4. **L^2-norm Relative Error** (*L2RE*):
L2RE provides a normalized measure of the L^2 error by dividing it by the norm of the actual values. The formula is:
$$L2RE(u, \hat{u}) = \frac{\|u - \hat{u}\|_{L^2(\Omega)}}{\|u\|_{L^2(\Omega)}}.$$

5. **H^1-norm Error** (*H1E*):
H1E is the error related to the Hilbert space $H^1(\Omega)$ and it is defined by the semi-norm in $H^1(\Omega)$ of the difference between the solutions as functions. This error take into account the difference in terms of the derivatives of the functions considered. So, this norm not only measures the error in the function

values but also incorporates the error in the gradient, thus providing a more comprehensive assessment of the approximation quality. The formula is:

$$H1E(u, \hat{u}) = |u - \hat{u}|_{H^1(\Omega)} = \|\nabla u - \nabla \hat{u}\|_{L^2(\Omega)}.$$

6. **H^1-norm Relative Error** (*H1RE*):
   H1RE provides a normalized measure of the H^1 error by dividing it by the semi-norm of the actual values. The formula is:

$$H1RE(u, \hat{u}) = \frac{|u - \hat{u}|_{H^1(\Omega)}}{|u|_{H^1(\Omega)}}.$$

Each of these metrics provides different perspectives on model performance, and, depending on the context, we might choose one or a combination of these metrics to get a comprehensive view of our model's effectiveness.

4.6 Implementation

The programming language employed for this thesis is Python, which is currently the most widely used in DL application. Python has a rich ecosystem of libraries and frameworks specifically designed for DL, such as TensorFlow and Keras, which are the ones we have used. These libraries provide high-level APIs that make it easier to build, train, and deploy DL models.
To execute the simulations, the primary platform utilized is HPC@DISMA. It is an academic computing center that provides computational resources and technical support for research activities for academic and didactic purposes. The HPC project is officially managed by Professor Matteo Cicuttin of the Department of Mathematical Sciences (DISMA).
The results of the experiments have been analyzed using some simple Jupyter notebooks on the widely used platform, Visual Studio Code.

5 Experimental Results

In this chapter, we present the outcomes of our experiments, which are divided based on the two problems addressed, *Test 1* and *Test 2*. As described in detail in Section 4.1, both problems are characterized by a specific set of parameters that determine their physical and geometric properties. The first is a fairly standard PDEs problem, while the second is purposely designed to be more complex in order to test the GINN on more difficult and challenging PDEs problems.
The analyses were conducted using all the errors described in Section 4.5; however, for the sake of clarity, we will focus mainly on the Mean Absolute Error (MAE) and, at times, on the Mean Relative Range Error (MRRE). This choice is due to the fact that, although the associated values change, the qualitative behavior of the various errors remains very similar.

5.1 Test 1

To begin, we present the general results of the experiments comparing FCNNs with GINNs for the first problem, *Test 1*. Figure 5.1 shows the average MAE and the average MRRE on the test set for both the architectures, divided according to the number of ground-truth simulations used for the training. We can observe that the GINNs consistently outperform the FCNNs; in fact, looking at the Figure 5.1, we notice that the average error of the GINNs is always lower than that of the FCNNs. As expected, performance improves for both the architectures as the number of training simulations increases, except for the FCNNs with a train set of 1024 elements. In particular, GINN architecture prove to be robust, as there is not a significant difference between the simulations with 1024 training data points and those with 512. The performance only visibly decreases when the training data is very limited, that is, below 256 examples.

For completeness, we also provide summary tables of all the experiments conducted on the first problem. Specifically, in Tables 5.1, 5.2, 5.3 and 5.4, we present the experiments performed on the FCNNs, while in Tables 5.5, 5.6, 5.7 and 5.8, those conducted on the GINNs. The tables are organized based on the number of ground truth simulations used during training and validation (the number of simulations used for testing the model is fixed at 3000). In these tables, we report the number of parameters used for training (n_params) along with some of the errors defined

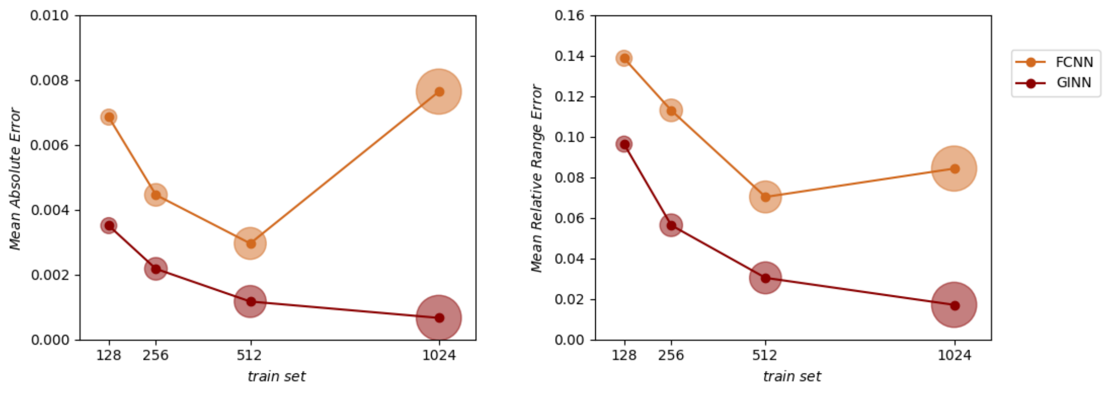


Figure 5.1: Comparison between FCNNs and GINNs on MAE and MRRE.

in Section 4.5, in particular, MAE, MRRE, L2E, and H1E. The data are sorted in ascending order of MAE.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	9657461	0.000353	0.004464	0.000360	0.031195
2	9657461	0.000847	0.025055	0.000806	0.064044
3	6508661	0.000904	0.013598	0.000964	0.080452
4	36090997	0.000912	0.027108	0.000845	0.074520
5	27698293	0.000984	0.038724	0.000868	0.076634
6	23501941	0.001037	0.021556	0.001091	0.091334
7	36090997	0.001063	0.019882	0.001091	0.088824
8	27698293	0.001072	0.020457	0.001045	0.080284
9	36115573	0.001458	0.034632	0.001483	0.110416
10	8607861	0.001528	0.024064	0.001602	0.134670
11	27714677	0.001580	0.030059	0.001575	0.108891
12	6508661	0.001775	0.048296	0.001769	0.144299
13	7566453	0.002120	0.070653	0.002030	0.135667
14	31894645	0.002445	0.040154	0.002479	0.210579
15	23501941	0.002524	0.061392	0.002504	0.189663
16	8607861	0.002609	0.052935	0.002595	0.216571
17	23534709	0.002921	0.103319	0.002956	0.222262
18	31943797	0.002940	0.119633	0.002906	0.230998
19	8632437	0.003032	0.205583	0.002963	0.210080
20	31894645	0.003037	0.088809	0.002937	0.269264
21	36115573	0.003131	0.087219	0.002940	0.227294
22	23534709	0.003207	0.133682	0.003009	0.242691
23	6525045	0.003208	0.093924	0.003174	0.230806
24	6525045	0.003284	0.073073	0.003094	0.244204
25	8632437	0.003672	0.145309	0.003695	0.271125
26	27714677	0.004777	0.168495	0.004432	0.360344
27	31943797	0.012181	0.153907	0.012018	0.688410

Continued on next page

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
28	9669749	0.050202	0.184287	0.049540	2.418116
29	7558261	0.102527	0.351533	0.094817	3.769222
30	7558261	0.252794	0.870350	0.237960	9.618701
31	9669749	1.095436	5.782240	1.025747	41.282606
32	7566453	2.754935	9.609385	2.587270	107.597768

Table 5.1: *Test 1*: Experiments executed on FCNN with 1024 training data and 512 validation data.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	27698293	0.000426	0.004615	0.000424	0.036083
2	36090997	0.000536	0.009791	0.000525	0.044062
3	9657461	0.000598	0.010086	0.000577	0.047575
4	7566453	0.001058	0.018321	0.001048	0.074543
5	23501941	0.001510	0.024923	0.001583	0.122557
6	9669749	0.001539	0.027580	0.001538	0.109290
7	27698293	0.001565	0.037887	0.001534	0.130583
8	9669749	0.001609	0.037064	0.001578	0.117807
9	27714677	0.001727	0.045959	0.001748	0.127743
10	36115573	0.001751	0.051480	0.001756	0.134820
11	6508661	0.001774	0.052174	0.001771	0.136862
12	9657461	0.002117	0.052961	0.002022	0.156980
13	8607861	0.002144	0.043105	0.002135	0.175621
14	36090997	0.002228	0.041818	0.002197	0.172878
15	8607861	0.002447	0.036736	0.002482	0.224850
16	31894645	0.002624	0.046412	0.002629	0.235299
17	6508661	0.002629	0.062408	0.002581	0.206788
18	23501941	0.002640	0.061312	0.002599	0.222776
19	23534709	0.003138	0.092587	0.003151	0.239347
20	31943797	0.003949	0.108069	0.003989	0.319406
21	6525045	0.004079	0.110325	0.003986	0.299401
22	36115573	0.004135	0.139319	0.003803	0.308549
23	6525045	0.004151	0.096216	0.003990	0.318911
24	27714677	0.004418	0.161096	0.004059	0.318956
25	23534709	0.004493	0.102318	0.004354	0.328214
26	8632437	0.004557	0.158580	0.004530	0.361685
27	31943797	0.004802	0.128117	0.004549	0.384061
28	8632437	0.004865	0.144902	0.004574	0.381902
29	7566453	0.005241	0.117824	0.004981	0.295314
30	7558261	0.005494	0.047610	0.005144	0.261375
31	31894645	0.007424	0.105121	0.007009	0.639236
32	7558261	6.410047	21.882278	5.322680	490.244137

Table 5.2: *Test 1*: Experiments executed on FCNN with 512 training data and 256 validation data.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	36090997	0.001548	0.022322	0.001509	0.118035
2	27698293	0.001958	0.027626	0.001907	0.127263
3	9657461	0.002341	0.039028	0.002286	0.154796
4	36115573	0.002803	0.061388	0.002759	0.221864
5	7558261	0.002848	0.082009	0.002740	0.196922
6	27714677	0.002929	0.059857	0.002863	0.231201
7	27698293	0.003127	0.094833	0.002901	0.228529
8	9669749	0.003148	0.060190	0.003071	0.242108
9	36090997	0.003201	0.088187	0.003117	0.248485
10	9657461	0.003334	0.075111	0.003233	0.187533
11	7566453	0.003337	0.068569	0.003302	0.245866
12	23501941	0.003629	0.075580	0.003552	0.292401
13	8607861	0.003664	0.058376	0.003654	0.286742
14	8607861	0.003774	0.056412	0.003706	0.301643
15	31894645	0.003849	0.064528	0.003731	0.291688
16	6508661	0.004133	0.102438	0.003997	0.311314
17	6508661	0.004218	0.109621	0.004060	0.266928
18	6525045	0.004592	0.097066	0.004331	0.343083
19	23501941	0.004692	0.110206	0.004525	0.306705
20	7566453	0.004711	0.165899	0.004424	0.294435
21	9669749	0.004751	0.163286	0.004357	0.342339
22	23534709	0.004890	0.134111	0.004829	0.369405
23	31943797	0.004983	0.116206	0.004996	0.407653
24	8632437	0.005031	0.078316	0.004987	0.402799
25	27714677	0.005190	0.147721	0.004786	0.375521
26	31943797	0.005338	0.150122	0.005210	0.422175
27	6525045	0.005403	0.145673	0.005040	0.392047
28	8632437	0.005885	0.151241	0.005620	0.451230
29	36115573	0.005935	0.197160	0.005624	0.408313
30	23534709	0.006387	0.181371	0.005986	0.476715
31	31894645	0.008084	0.181578	0.007325	0.663728
32	7558261	0.012746	0.446227	0.011683	0.850779

Table 5.3: *Test 1*: Experiments executed on FCNN with 256 training data and 128 validation data.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	36115573	0.004348	0.065199	0.004122	0.329426
2	27698293	0.004496	0.072811	0.004343	0.254427
3	27714677	0.004583	0.073597	0.004389	0.359078
4	9657461	0.004636	0.090978	0.004519	0.294440
5	36090997	0.004818	0.081713	0.004661	0.315815
6	6508661	0.004907	0.044822	0.004834	0.287990
7	9669749	0.005136	0.114453	0.004706	0.383426
8	7566453	0.005204	0.091724	0.004721	0.391601
9	7558261	0.005225	0.104568	0.004918	0.307289

Continued on next page

	n_params	MAE	MRRE	L2E	H1E
10	7558261	0.005305	0.118322	0.005113	0.331271
11	6525045	0.005643	0.109577	0.005395	0.422717
12	9657461	0.005697	0.155536	0.005351	0.341094
13	7566453	0.005713	0.218801	0.005325	0.421031
14	6525045	0.005850	0.135141	0.005597	0.416060
15	36090997	0.005963	0.130321	0.005460	0.442856
16	23501941	0.005999	0.142622	0.005781	0.380374
17	8632437	0.006077	0.093102	0.006042	0.460152
18	31943797	0.006166	0.139514	0.006071	0.492704
19	8632437	0.006310	0.136386	0.006235	0.485498
20	9669749	0.006385	0.154918	0.005978	0.461333
21	23534709	0.006464	0.091446	0.006342	0.479647
22	8607861	0.006524	0.114655	0.006427	0.423542
23	36115573	0.006692	0.147320	0.006315	0.481226
24	27698293	0.007086	0.110521	0.006858	0.462353
25	27714677	0.007264	0.259223	0.006839	0.493661
26	31894645	0.007269	0.085655	0.007157	0.554968
27	23534709	0.007413	0.226558	0.006969	0.555864
28	31894645	0.007694	0.113763	0.007369	0.603823
29	31943797	0.007888	0.254340	0.007369	0.586037
30	23501941	0.008715	0.124320	0.008194	0.654441
31	8607861	0.013126	0.195184	0.013055	0.873833
32	6508661	0.024480	0.435707	0.022556	1.470933

Table 5.4: *Test 1*: Experiments executed on FCNN with 128 training data and 64 validation data.

	n_params	MAE	MRRE	L2E	H1E
1	1900389	0.000323	0.007260	0.000297	0.023334
2	1423291	0.000385	0.009877	0.000356	0.027344
3	1888979	0.000393	0.010905	0.000363	0.028533
4	1434701	0.000403	0.010672	0.000368	0.028382
5	1866159	0.000427	0.011111	0.000397	0.030980
6	1888355	0.000467	0.011721	0.000435	0.033213
7	1900389	0.000468	0.010848	0.000435	0.035298
8	1877569	0.000490	0.013754	0.000456	0.035396
9	1888979	0.000491	0.011179	0.000452	0.036232
10	1421009	0.000496	0.013217	0.000460	0.035350
11	1434237	0.000540	0.013752	0.000504	0.038420
12	1865535	0.000585	0.015044	0.000549	0.042052
13	1876945	0.000596	0.015871	0.000551	0.042474
14	1899765	0.000606	0.015267	0.000560	0.043452
15	1888355	0.000630	0.017703	0.000583	0.045149
16	1899765	0.000636	0.017754	0.000590	0.046527
17	1420545	0.000641	0.017218	0.000590	0.046211
18	1409599	0.000642	0.018917	0.000606	0.048363

Continued on next page

	n_params	MAE	MRRE	L2E	H1E
19	1409135	0.000712	0.019846	0.000665	0.051553
20	1877569	0.000733	0.019278	0.000692	0.054574
21	1434701	0.000737	0.022681	0.000706	0.057423
22	1422827	0.000740	0.019772	0.000681	0.053675
23	1409599	0.000760	0.016750	0.000728	0.054947
24	1865535	0.000810	0.019528	0.000763	0.059614
25	1420545	0.000840	0.018622	0.000802	0.060907
26	1876945	0.000849	0.018708	0.000817	0.062791
27	1866159	0.000859	0.018328	0.000842	0.066333
28	1423291	0.000879	0.021180	0.000835	0.063756
29	1422827	0.000884	0.022889	0.000839	0.063361
30	1421009	0.001037	0.026862	0.000997	0.078055
31	1409135	0.001049	0.025267	0.000998	0.077119
32	1434237	0.001148	0.034136	0.001101	0.087169

Table 5.5: *Test 1*: Experiments executed on GINN with 1024 training data and 512 validation data.

	n_params	MAE	MRRE	L2E	H1E
1	1900389	0.000681	0.017136	0.000635	0.047507
2	1888979	0.000732	0.017801	0.000685	0.052597
3	1888979	0.000811	0.020328	0.000758	0.059217
4	1421009	0.000818	0.021429	0.000764	0.059828
5	1434701	0.000818	0.019977	0.000770	0.059097
6	1423291	0.000826	0.017219	0.000787	0.059537
7	1900389	0.000837	0.021913	0.000795	0.061000
8	1866159	0.000903	0.024284	0.000836	0.063851
9	1409599	0.000912	0.021599	0.000881	0.067159
10	1877569	0.000948	0.026571	0.000897	0.070415
11	1899765	0.001000	0.022937	0.000945	0.070916
12	1420545	0.001067	0.029323	0.001017	0.079418
13	1434701	0.001068	0.027064	0.001011	0.077754
14	1865535	0.001069	0.029930	0.001021	0.078755
15	1888355	0.001115	0.031987	0.001049	0.082933
16	1409135	0.001118	0.034527	0.001040	0.082340
17	1422827	0.001138	0.033941	0.001071	0.083159
18	1899765	0.001139	0.031274	0.001073	0.085321
19	1888355	0.001189	0.036782	0.001127	0.088070
20	1876945	0.001201	0.033401	0.001115	0.087410
21	1865535	0.001303	0.029938	0.001235	0.096768
22	1434237	0.001344	0.038058	0.001255	0.100653
23	1421009	0.001405	0.028519	0.001383	0.106758
24	1866159	0.001408	0.031416	0.001375	0.108064
25	1423291	0.001446	0.036917	0.001415	0.108479
26	1409599	0.001473	0.034953	0.001421	0.108566
27	1876945	0.001491	0.034196	0.001448	0.113935

Continued on next page

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
28	1420545	0.001499	0.031528	0.001455	0.113281
29	1434237	0.001529	0.042972	0.001449	0.111805
30	1877569	0.001535	0.030288	0.001504	0.115284
31	1409135	0.001725	0.055586	0.001636	0.130589
32	1422827	0.001841	0.057302	0.001725	0.136434

Table 5.6: *Test 1*: Experiments executed on GINN with 512 training data and 256 validation data.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	1900389	0.001549	0.034815	0.001473	0.113485
2	1434701	0.001605	0.035520	0.001516	0.117943
3	1888979	0.001631	0.042344	0.001560	0.120869
4	1866159	0.001637	0.036129	0.001588	0.121808
5	1888979	0.001649	0.041155	0.001568	0.125152
6	1421009	0.001829	0.043416	0.001761	0.137393
7	1899765	0.001841	0.045662	0.001753	0.136887
8	1865535	0.001853	0.047784	0.001763	0.141284
9	1900389	0.001897	0.045824	0.001850	0.142326
10	1876945	0.001897	0.044875	0.001791	0.138943
11	1899765	0.001901	0.046314	0.001795	0.142622
12	1423291	0.001904	0.048306	0.001788	0.139115
13	1888355	0.001909	0.042616	0.001841	0.140623
14	1422827	0.001921	0.052957	0.001822	0.142111
15	1434701	0.001961	0.055181	0.001857	0.149980
16	1877569	0.001965	0.046949	0.001845	0.144631
17	1409599	0.001967	0.045896	0.001869	0.143936
18	1420545	0.002029	0.057573	0.001894	0.150602
19	1888355	0.002031	0.052244	0.001926	0.152512
20	1434237	0.002039	0.057794	0.001907	0.150670
21	1423291	0.002163	0.059020	0.002059	0.164629
22	1409135	0.002190	0.060213	0.002058	0.163844
23	1421009	0.002222	0.057150	0.002113	0.169588
24	1865535	0.002572	0.078188	0.002413	0.194958
25	1409135	0.002597	0.076085	0.002451	0.195244
26	1434237	0.002717	0.075796	0.002553	0.207480
27	1422827	0.002732	0.078424	0.002592	0.208113
28	1876945	0.002749	0.070116	0.002617	0.211843
29	1866159	0.002791	0.072355	0.002686	0.217131
30	1420545	0.002945	0.076912	0.002773	0.223159
31	1877569	0.003044	0.068784	0.002976	0.243568
32	1409599	0.003851	0.104589	0.003713	0.310127

Table 5.7: *Test 1*: Experiments executed on GINN with 256 training data and 128 validation data.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	1900389	0.002441	0.060848	0.002388	0.185966
2	1866159	0.002561	0.053482	0.002493	0.193338
3	1900389	0.002612	0.055455	0.002506	0.195830
4	1888979	0.002616	0.052937	0.002534	0.198230
5	1888979	0.002705	0.074787	0.002599	0.198698
6	1434701	0.002789	0.062193	0.002692	0.208865
7	1877569	0.002837	0.070943	0.002723	0.214547
8	1423291	0.003007	0.076848	0.002858	0.223904
9	1423291	0.003050	0.078783	0.002900	0.227096
10	1421009	0.003060	0.072202	0.002932	0.234422
11	1409599	0.003205	0.074762	0.003068	0.242995
12	1876945	0.003237	0.087109	0.003051	0.241565
13	1434701	0.003263	0.084893	0.003055	0.251136
14	1888355	0.003268	0.099716	0.003105	0.246290
15	1409135	0.003393	0.082421	0.003277	0.260308
16	1899765	0.003504	0.097361	0.003329	0.266251
17	1865535	0.003526	0.090750	0.003410	0.269221
18	1434237	0.003601	0.103887	0.003394	0.270545
19	1888355	0.003648	0.098100	0.003528	0.275070
20	1422827	0.003648	0.102458	0.003473	0.279828
21	1421009	0.003664	0.091444	0.003557	0.280545
22	1420545	0.003803	0.109323	0.003606	0.289851
23	1865535	0.003822	0.120056	0.003574	0.283405
24	1899765	0.003840	0.111305	0.003652	0.288444
25	1420545	0.003988	0.112221	0.003776	0.302259
26	1434237	0.004016	0.120256	0.003788	0.303152
27	1409135	0.004034	0.130822	0.003766	0.306300
28	1422827	0.004168	0.128064	0.003953	0.319402
29	1409599	0.004290	0.141558	0.004072	0.318944
30	1876945	0.004411	0.117619	0.004231	0.342679
31	1866159	0.004953	0.154559	0.004686	0.371741
32	1877569	0.005209	0.162763	0.004987	0.400427

Table 5.8: *Test 1*: Experiments executed on GINN with 128 training data and 64 validation data.

We would like to again point out to the reader that the GINNs are much lighter networks, as they require a significantly lower number of training parameters compared to the FCNNs. For the first problem, we observe an average of 18951285 parameters for training the FCNNs and an average of 1652440 parameters for training the GINNs, i.e., less than 9% compared to the average for the FCNNs.

5.1.1 Hyperparameter Analysis

As described in Section 4.4, an extensive grid search for hyperparameters has been performed, testing various values for both the FCNNs and the GINNs, as illustrated in Table 4.2. Each hyperparameter have a pair of values to test for each network architecture and the goal is to find, in general, the combination of hyperparameters that works best, so each pair of hyperparameters is analyzed individually. The results are presented below.

- *Number of Hidden Layers (nh)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the qualitative results are shown in Figure 5.2. In the case of FCNNs, $nh = 8$ performs slightly better than $nh = 6$ and in the case of GINNs, $nh = 80$ outperforms $nh = 60$ but, in both cases, the difference does not appear to be of substantial significance.

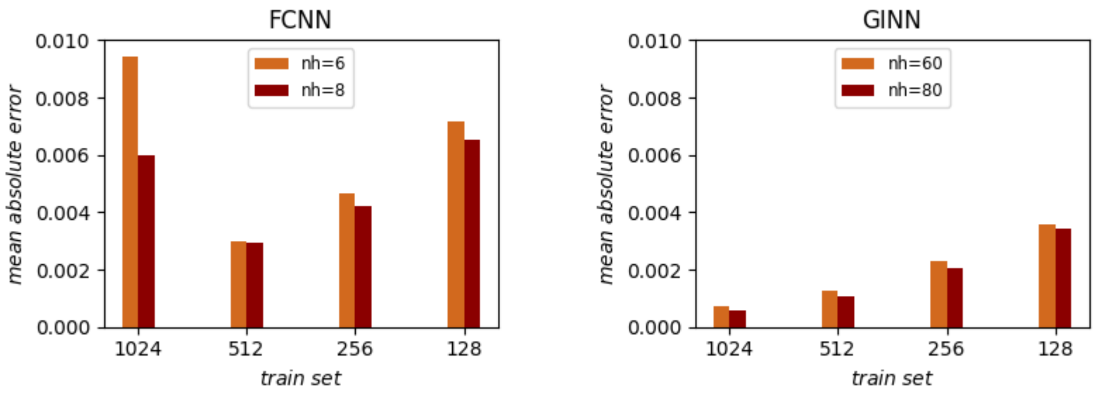


Figure 5.2: Number of Hidden Layers (nh).

- *Use of Residual connections (rs)*
  This hyperparameter was used exclusively for the FCNNs, and the qualitative results are shown in Figure 5.3. The use of residual blocks seems to help in cases with small training sets (256, 128), but appears to worsen the situation when a larger training set (1024, 512) is used.

- *Using Batch Normalization (bn)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the qualitative results are shown in Figure 5.4. In the case of FCNNs, there is no clear predominance in the use or non-use of batch normalization. Instead, in the case of GINNs, the use of batch normalization always seems to improve the model's performance.

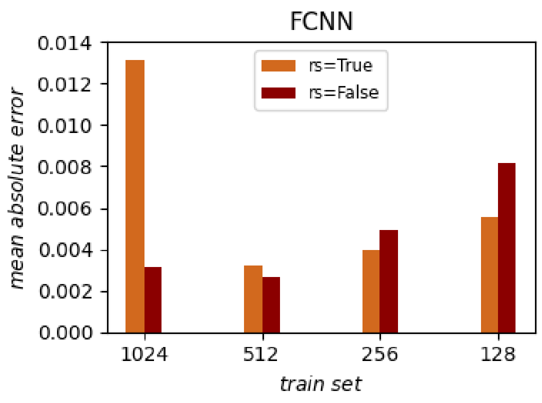


Figure 5.3: Use of Residual connections (rs).

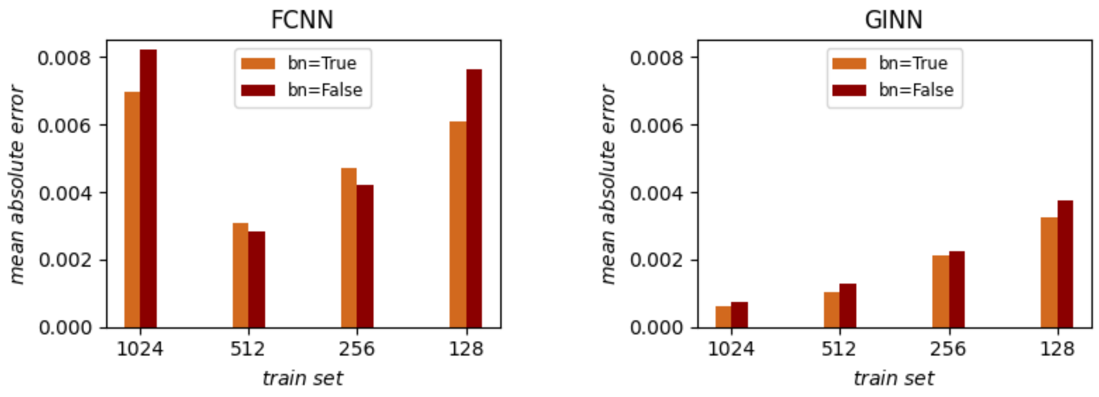


Figure 5.4: Using Batch Normalization (bn).

- *Activation Function (af)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the qualitative results are shown in Figure 5.5. In the case of FCNNs, the *mish* activation function almost always performs better. Instead, in the case of GINNs, the *elu* activation function performs always better.

- *Width of Hidden Layers (wh)*
  This hyperparameter was used exclusively for the FCNNs, and the qualitative results are shown in Figure 5.6. It seems that a layer with a width of 2048 performs better than the other, but the difference is not significant.

- *Number of Filters (nf)*
  This hyperparameter was used exclusively for the GINNs, and the qualitative results are shown in Figure 5.7. It seems that a layer with 5 filters performs

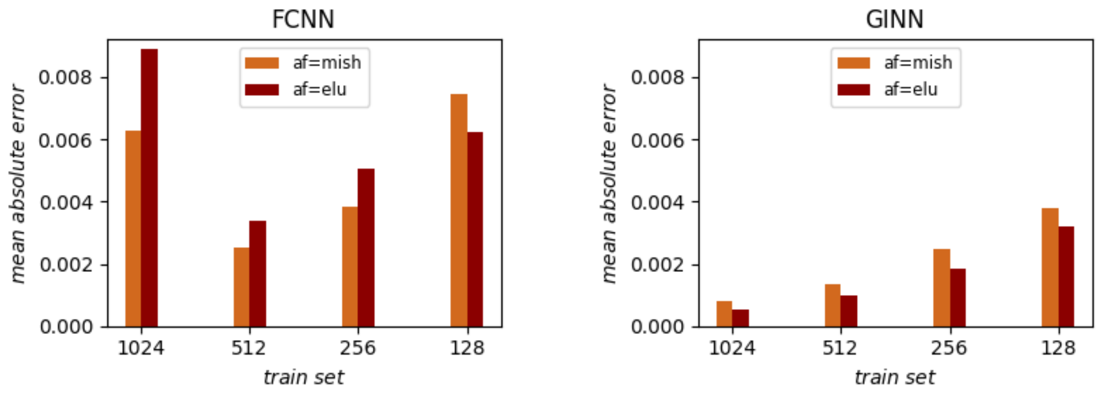


Figure 5.5: Activation Function (af).

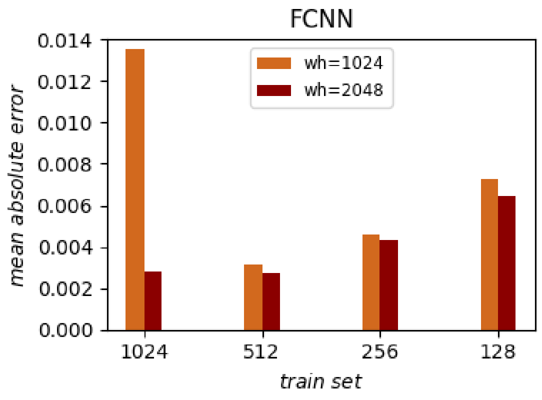


Figure 5.6: Width of Hidden Layers (wh).

better than the one with 3 but, even in this case, the difference is not significant.

- *Input Refresh (rf)*
  This hyperparameter was used exclusively for the GINNs, and the qualitative results are shown in Figure 5.8. This hyperparameter proves to be one of the most significant, as the benefit of refreshing the inputs is visually noticeable.

To conclude the analysis of the hyperparameters, we search for the ideal combinations for each type of network. We observed that FCNNs do not have an ideal configuration of hyperparameters, because no one results to be always more effective than its counterpart; nonetheless, we select as ideal parameters the following: the use of batch normalization, the width of the hidden layers of 2048, and the use of the *mish* activation function. GINNs, instead, show more consistent results,

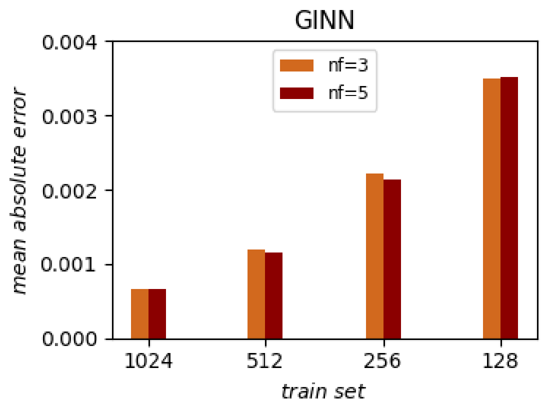


Figure 5.7: Number of Filters (nf).

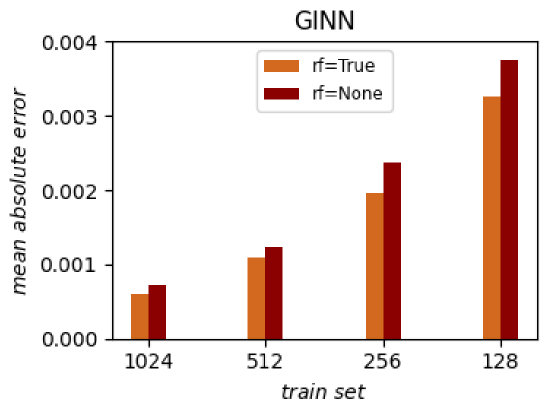


Figure 5.8: Input Refresh (rf).

with certain hyperparameters almost always performing better. Specifically, the following hyperparameters were identified as optimal: the use of batch normalization, the practice of refreshing the inputs every n layers (in our case, $n = 15$), and the use of the *elu* activation function.
In Figure 5.9 and Figure 5.10, all the MAEs related to the different configurations for both GINNs and FCNNs are shown, with emphasis on those related to the configurations that contain the hyperparameters combination we identified as the "best". We can see that for each case, there are multiple points highlighted as ideal. This is due to the fact that, out of the 5 hyperparameters we analyzed, only 3 were selected as determining parameters. From the previous study, we observed that some hyperparameters have very little influence, so the quality of the prediction is minimally or not at all affected by changes in these hyperparameters. However,

looking at the figures overall, we point out to the reader that for GINNs, the configurations highlighted as ideal are indeed among the models with the lowest error. This is not the case for FCNNs, which therefore appear to be less consistent.

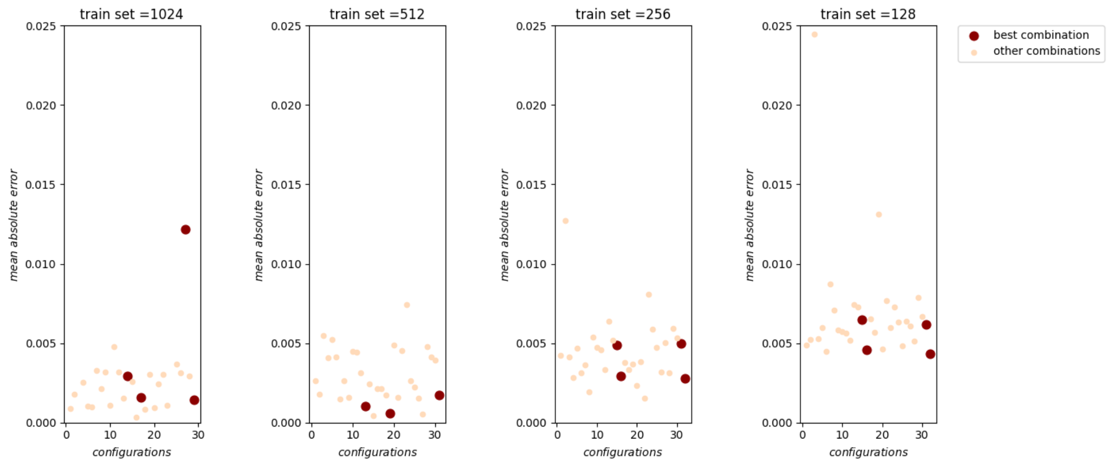


Figure 5.9: Best combination for FCNNs: use of batch normalization = True, width of hidden layers = 2048, activation function = *mish*.

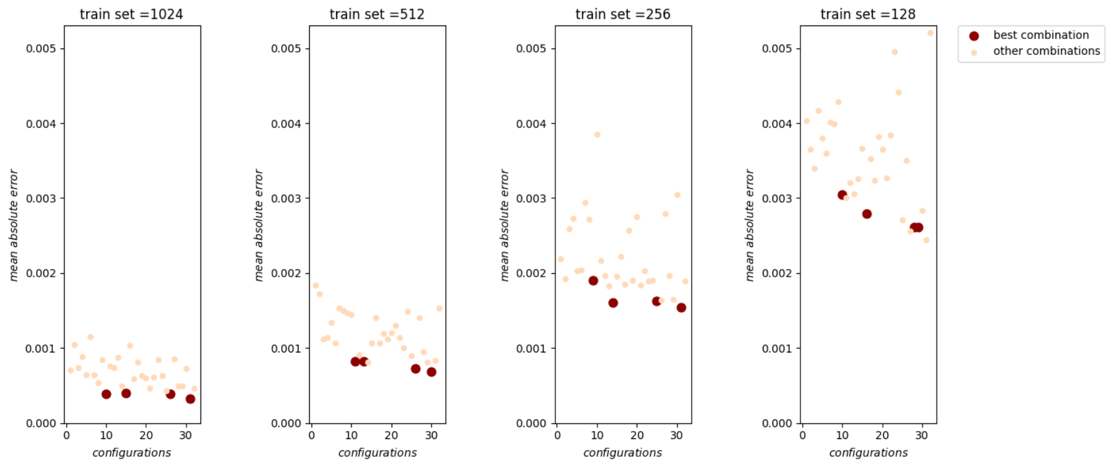


Figure 5.10: Best combination for GINNs: use of batch normalization = True, refresh input = True, activation function = *elu*.

5.1.2 Qualitative Analysis of the Best Model

About *Test 1*, the best-performing GINN configuration according to MAE is the one with: number of hidden layers = 80, use of batch normalization = True,

number of filters = 5, activation function = *elu* and input's refresh = True.
Let us conclude the section with a qualitative analysis by showing some cases from this configuration, comparing our model's predictions with the ground truth solutions. In particular, the two best cases, Figures 5.11 and 5.12, and two worst cases, Figures 5.17 and 5.18, with respect to MAE and the two best cases, Figures 5.13 and 5.14, and two worst cases, Figures 5.19 and 5.20, with respect the MRRE. In addition in Figures 5.15 and 5.16 we report the predictions from two random test cases.
By observing the figures, we can make some considerations. First of all, regarding the MRRE, we can say that the best predictions are those that show on the inner circle a single window where the Neumann condition has an high value ($\mu_c \to 1$) on either the right or left side of the circle. On the contrary, the worst predictions seem to be those in which we have a diffusion that is almost completely homogeneous and close to zero. As for the MAE, we identify the best predictions as those where the entire circumference has a very low Neumann value ($\mu_c \to 0$). Conversely, we identify the worst predictions as those that have a significant portion of the circle, more than half, where $\mu_c \to 1$. Among the "random" cases, we have one that shows two arcs of the circumference where $\mu_c \to 1$ on opposite sides of the circle, and another that presents only a couple of points where Neumann has a high value.

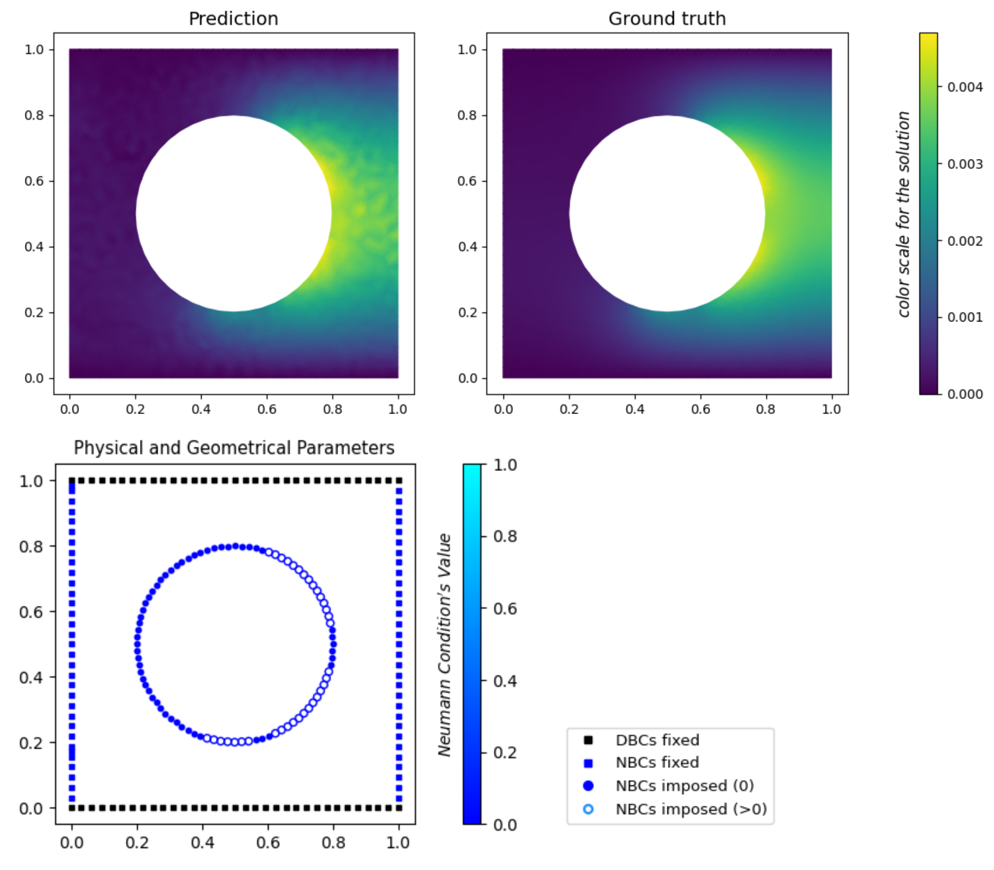


Figure 5.11: Best prediction with respect to MAE, $MAE = 5.6068e - 05$ and $MRRE = 0.01223$.

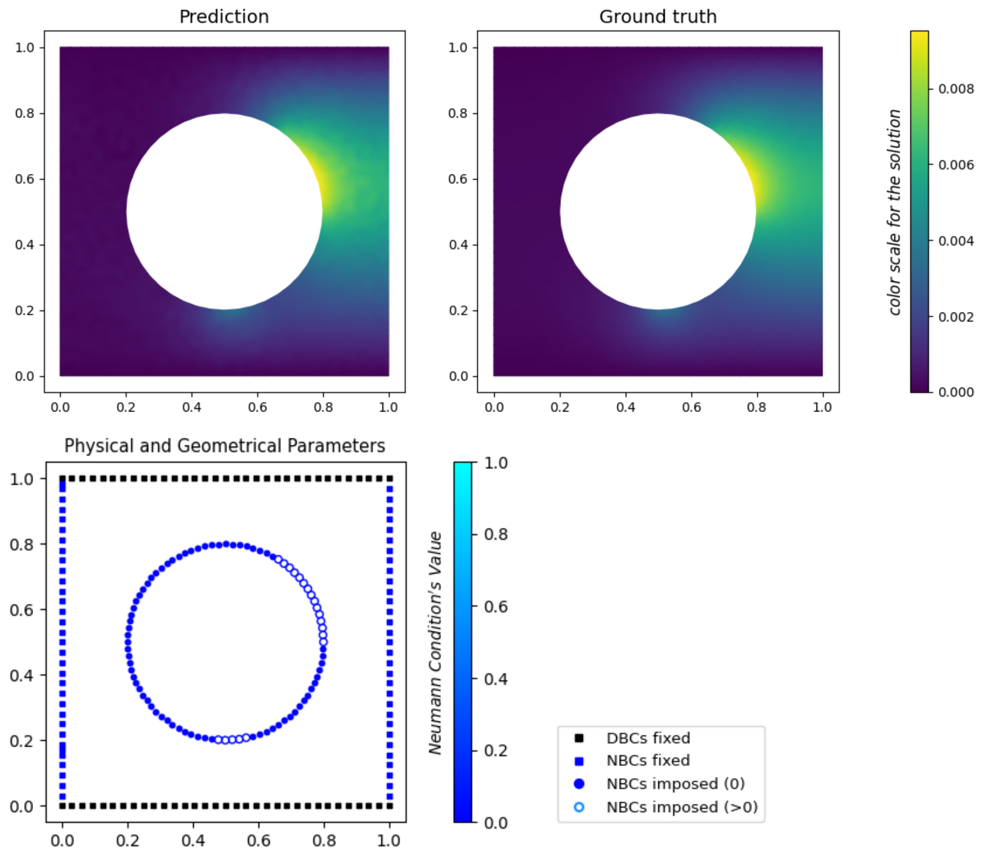


Figure 5.12: Second best prediction with respect to MAE, $MAE = 5.6155e-05$ and $MRRE = 0.00592$.

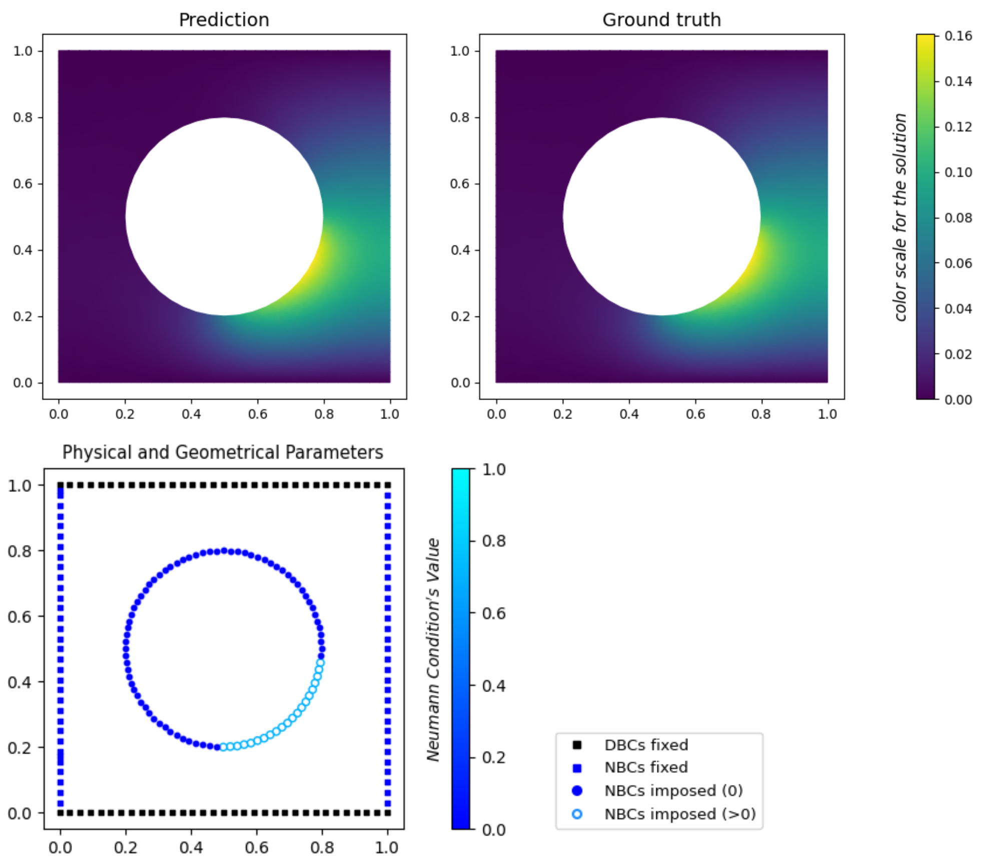


Figure 5.13: Best prediction with respect to MRRE, $MAE = 0.000168$ and $MRRE = 0.00105$.

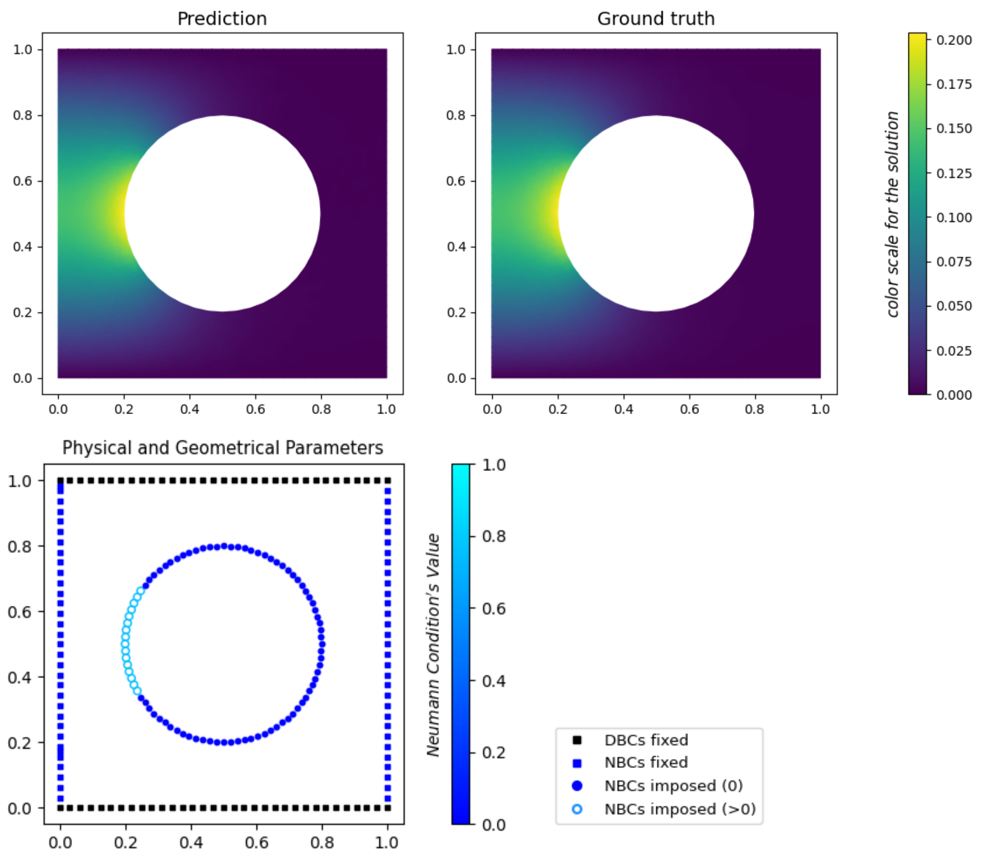


Figure 5.14: Second best prediction with respect to MRRE, $MAE = 0.00021$ and $MRRE = 0.00107$.

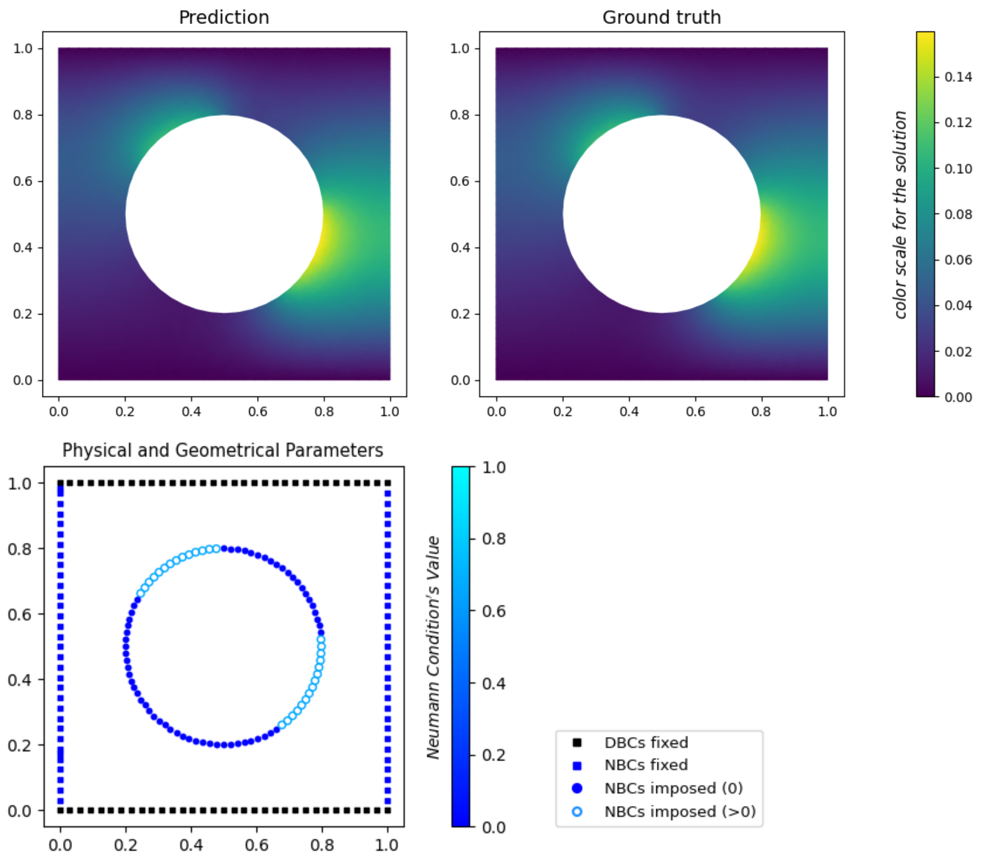


Figure 5.15: A random prediction, $MAE = 0.00028$ and $MRRE = 0.00178$.

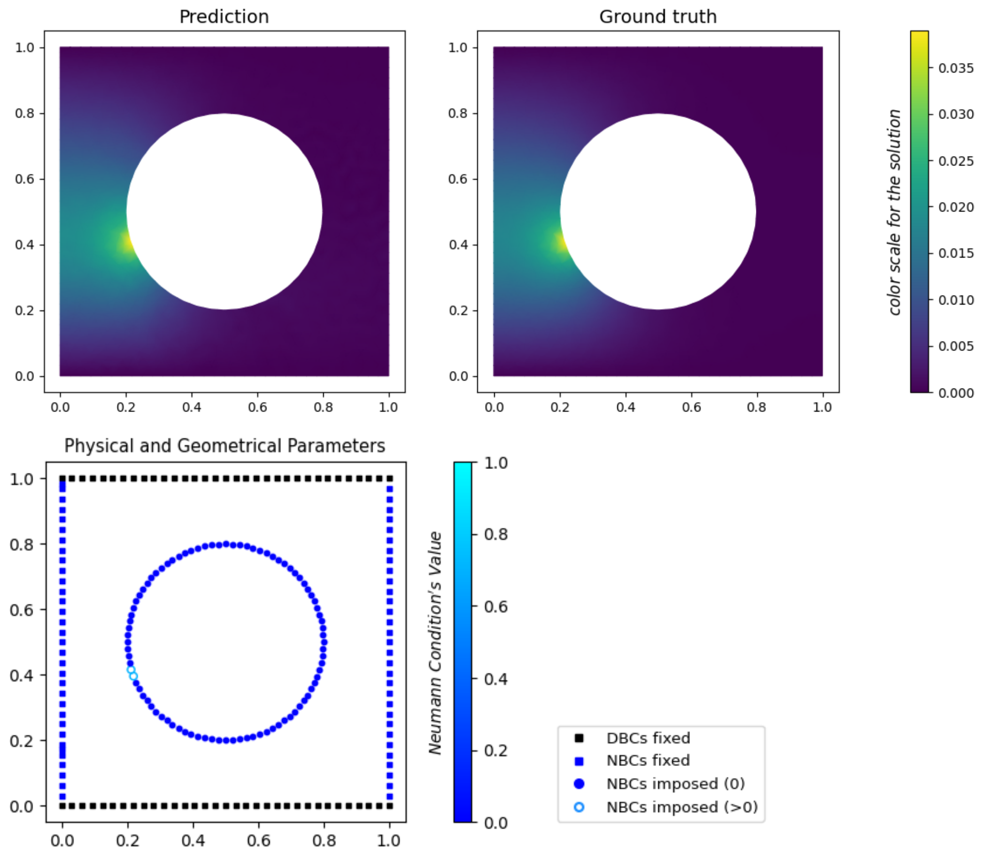


Figure 5.16: A second random prediction, $MAE = 0.00041$ and $MRRE = 0.01080$.

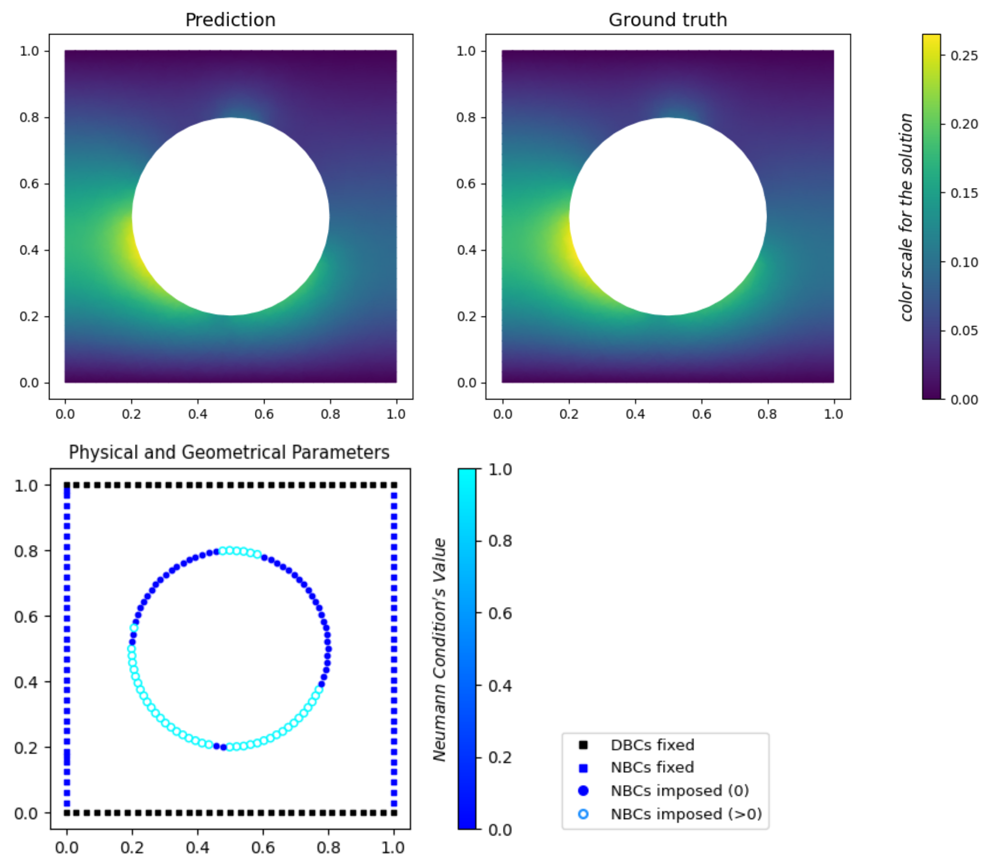


Figure 5.17: Worst prediction with respect to MAE, $MAE = 0.00300$ and $MRRE = 0.01133$.

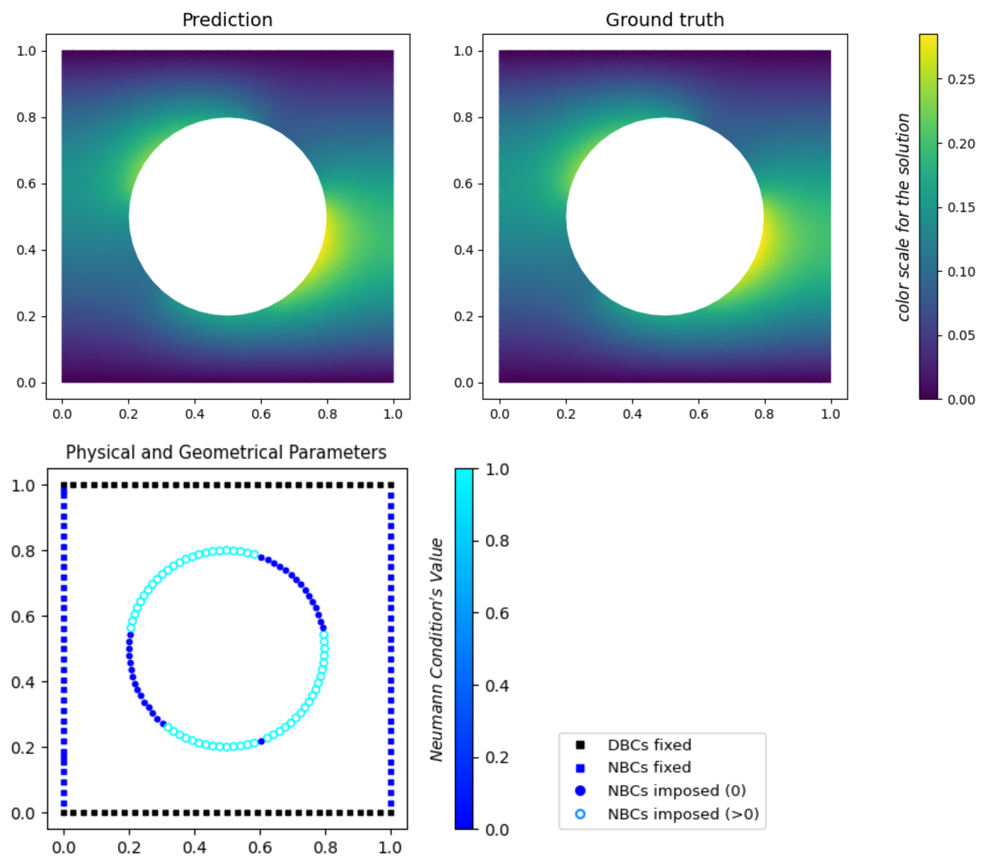


Figure 5.18: Second worst prediction with respect to MAE, $MAE = 0.00268$ and $MRRE = 0.00942$.

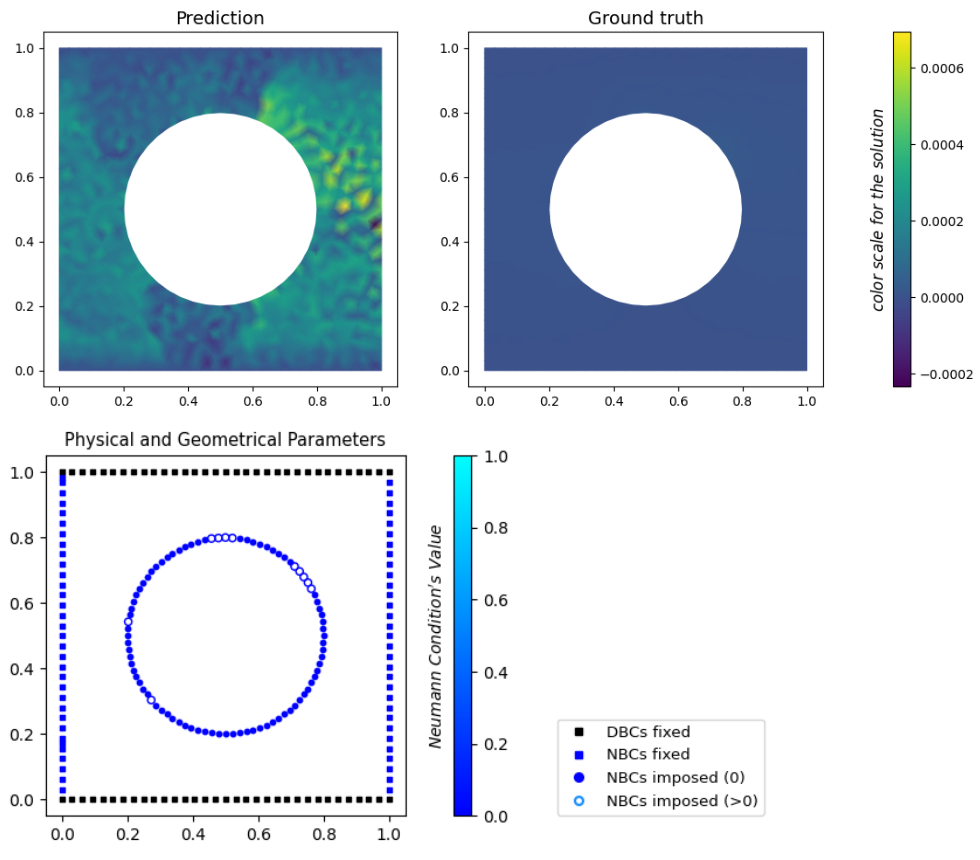


Figure 5.19: Worst prediction with respect to MRRE, $MAE = 0.00014$ and $MRRE = 5.07227$.

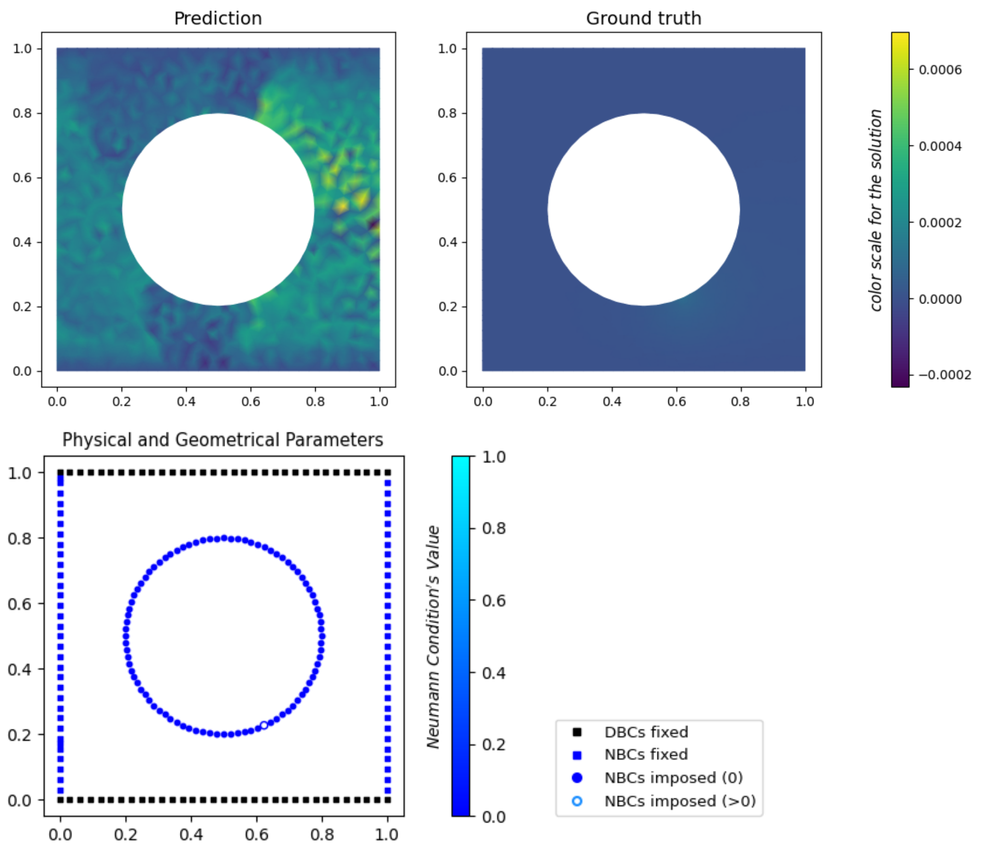


Figure 5.20: Second worst prediction with respect to MRRE, $MAE = 0.00014$ and $MRRE = 1.39954$.

5.2 Test 2

As for the first problem, we begin by presenting the general results of the experiments comparing FCNNs with GINNs for the second problem, *Test 2*. Figure 5.21 shows the average MAE and the average MRRE on the test set for both the architectures, divided according to the number of ground-truth simulations used for the training. We can observe that the GINNs consistently outperforms the FCNNs; in fact, looking at the Figure 5.21, we notice that the average error of the GINNs is always lower than that of the FCNNs. The architectures appear to be robust, as there is not a significant difference between the simulations with 1024 training data points and those with 512.

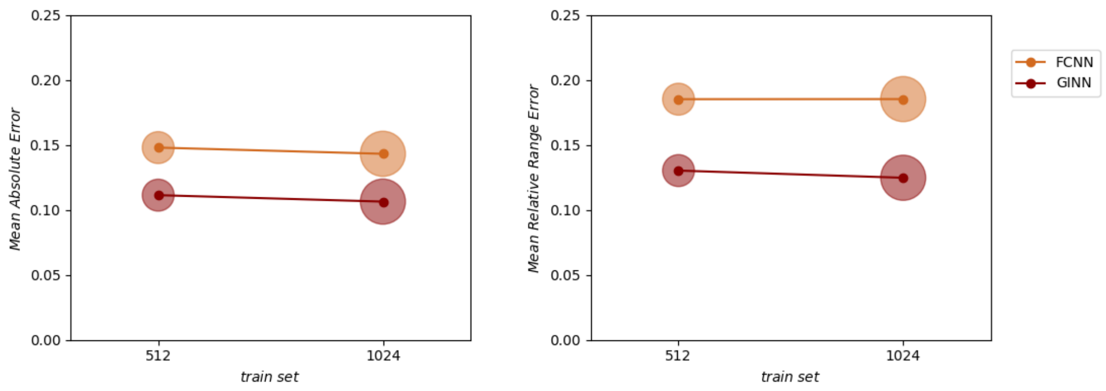


Figure 5.21: Comparison between FCNNs and GINNs on MAE and MRRE.

For completeness, we also provide summary tables of all the experiments conducted on the second problem. Specifically, in Tables 5.9 and 5.10, we present the experiments performed on the FCNNs, while in Tables 5.11 and 5.12, those conducted on the GINNs. The tables are organized based on the number of ground truth simulations used during training and validation (the number of simulations used for testing the model is fixed at 1500). In these tables, we report the number of parameters used for training (n_params) along with some of the errors defined in Section 4.5, in particular, MAE, MRRE, L2E, and H1E. The data are sorted in ascending order of MAE.

	n_params	**MAE**	**MRRE**	**L2E**	**H1E**
1	33959807	0.075988	0.076851	0.086225	10.866670
2	38156159	0.086432	0.088946	0.097805	12.583630
3	38205311	0.093141	0.090850	0.105703	12.789157
4	135126911	0.093726	0.092467	0.105801	12.938055

Continued on next page

	n__params	**MAE**	**MRRE**	**L2E**	**H1E**
5	38205311	0.100123	0.103325	0.109535	13.675308
6	101531519	0.102337	0.106536	0.111941	13.641345
7	101531519	0.105128	0.114846	0.117758	13.556540
8	29796223	0.106323	0.120400	0.115787	13.906145
9	29763455	0.107889	0.109876	0.117488	15.289824
10	135126911	0.108690	0.114502	0.118461	14.422343
11	29796223	0.109150	0.121490	0.121530	13.801449
12	151859071	0.120740	0.132620	0.132802	15.616593
13	33976191	0.122616	0.131634	0.134003	14.990111
14	118280063	0.123674	0.128490	0.133129	15.476903
15	42377087	0.124882	0.119398	0.135311	15.393478
16	151809919	0.128348	0.140382	0.134432	17.484791
17	33976191	0.133292	0.150747	0.144687	16.120885
18	42377087	0.141713	0.156252	0.151861	17.370806
19	29763455	0.143996	0.216869	0.151635	18.281931
20	151859071	0.146972	0.177555	0.158206	18.002559
21	118280063	0.151400	0.215425	0.163001	18.253939
22	101465983	0.152076	0.272121	0.159462	21.115543
23	33959807	0.163842	0.163081	0.177435	19.191855
24	151809919	0.169418	0.167193	0.183496	20.678195
25	118247295	0.170075	0.184253	0.180587	21.007894
26	101465983	0.171203	0.153781	0.181695	20.792956
27	135028607	0.177234	0.167017	0.189383	20.761473
28	42352511	0.200703	0.255371	0.212715	22.719108
29	42352511	0.231882	0.431978	0.247850	24.993835
30	135028607	0.238346	0.472643	0.256928	25.039884
31	118247295	0.238502	0.470023	0.256874	25.021957
32	38156159	0.238724	0.477767	0.257441	25.072597

Table 5.9: *Test 2*: Experiments executed on FCNN with 1024 training data and 512 validation data.

	n__params	**MAE**	**MRRE**	**L2E**	**H1E**
1	33959807	0.085989	0.082663	0.098335	12.345798
2	38156159	0.088853	0.085926	0.102059	12.792202
3	42352511	0.090082	0.088943	0.099927	12.662187
4	118247295	0.093321	0.094328	0.103626	13.073547
5	29763455	0.097432	0.096338	0.109672	13.259286
6	29763455	0.102658	0.120030	0.112300	13.953442
7	38205311	0.119248	0.128178	0.131069	15.747518
8	38205311	0.119912	0.133658	0.134136	15.073346
9	29796223	0.121669	0.128095	0.135081	14.892812
10	101531519	0.124289	0.130350	0.136420	15.393205
11	151809919	0.128218	0.138631	0.137166	18.006221
12	135126911	0.130706	0.162498	0.142990	16.906062
13	101531519	0.131534	0.161568	0.143037	16.355733

Continued on next page

	n_params	MAE	MRRE	L2E	H1E
14	29796223	0.132043	0.140906	0.142889	16.756880
15	135126911	0.137591	0.161496	0.148414	17.648015
16	151809919	0.145014	0.148716	0.158007	18.073982
17	42377087	0.146037	0.167100	0.160307	16.922287
18	151859071	0.146201	0.169963	0.158948	17.367668
19	42377087	0.147104	0.198210	0.161191	17.807039
20	33976191	0.153807	0.203566	0.168885	17.602801
21	33976191	0.161920	0.189015	0.174318	19.080601
22	118280063	0.164669	0.221874	0.178965	19.450996
23	118280063	0.167118	0.208582	0.183809	18.887331
24	135028607	0.175669	0.192828	0.185455	20.508727
25	118247295	0.177235	0.183867	0.191272	20.937997
26	101465983	0.181341	0.186741	0.196476	21.656276
27	135028607	0.183366	0.197452	0.197691	20.862528
28	33959807	0.183630	0.190908	0.196121	21.561967
29	151859071	0.188242	0.244341	0.198822	21.984333
30	42352511	0.229910	0.403972	0.247823	24.673764
31	101465983	0.236849	0.466503	0.254190	25.022207
32	38156159	0.239778	0.495001	0.259648	25.565102

Table 5.10: *Test 2*: Experiments executed on FCNN with 512 training data and 256 validation data.

	n_params	MAE	MRRE	L2E	H1E
1	6541583	0.081277	0.081663	0.099370	13.231994
2	6542207	0.082591	0.083319	0.104016	13.212254
3	8128383	0.085487	0.085502	0.109074	14.077067
4	6700887	0.085824	0.086479	0.107757	13.618085
5	8129167	0.086562	0.084925	0.106641	13.597468
6	8418758	0.090768	0.093357	0.110488	13.823480
7	8228342	0.091331	0.095796	0.109444	13.645281
8	6800062	0.092988	0.098448	0.112913	13.887477
9	6640758	0.093117	0.093988	0.114654	14.625740
10	6799438	0.093755	0.096501	0.115501	14.483825
11	6641382	0.095548	0.100732	0.113679	13.938235
12	8227558	0.096746	0.098854	0.115462	14.640278
13	8319583	0.096948	0.101619	0.113874	13.883945
14	6799438	0.097462	0.103489	0.116908	14.412815
15	6541583	0.097542	0.099333	0.117353	14.784137
16	8417974	0.097815	0.102319	0.115905	14.380196
17	6640758	0.098209	0.115575	0.115158	15.345822
18	6700263	0.098331	0.097349	0.118261	14.678742
19	8129167	0.099059	0.110046	0.119083	14.440806
20	8318799	0.099591	0.103904	0.117984	14.522970
21	6700263	0.100132	0.103192	0.123244	15.247715
22	6700887	0.100946	0.104842	0.122384	14.731367

Continued on next page

	n_params	MAE	MRRE	L2E	H1E
23	6800062	0.101429	0.101964	0.122153	14.770999
24	6542207	0.107186	0.114678	0.125746	15.350445
25	8318799	0.107980	0.112798	0.128113	15.421922
26	6641382	0.108563	0.120653	0.129795	15.789471
27	8418758	0.108781	0.112037	0.128269	15.401448
28	8228342	0.108853	0.112968	0.127756	15.631648
29	8319583	0.109898	0.114940	0.128635	15.430408
30	8417974	0.110988	0.114898	0.129630	15.638453
31	8227558	0.238244	0.471265	0.256628	25.009660
32	8128383	0.238323	0.471305	0.256676	25.011890

Table 5.11: *Test 2*: Experiments executed on GINN with 1024 training data and 512 validation data.

	n_params	MAE	MRRE	L2E	H1E
1	8228342	0.087901	0.083003	0.110174	13.866183
2	8129167	0.090787	0.101272	0.112003	13.996825
3	6542207	0.092082	0.094081	0.112940	14.190089
4	6800062	0.093158	0.095256	0.114754	14.270198
5	8128383	0.093621	0.088796	0.118643	15.131455
6	8227558	0.093977	0.094896	0.118581	15.082644
7	6641382	0.094375	0.092954	0.117294	14.659817
8	6700887	0.095644	0.096370	0.116597	14.467736
9	6700263	0.096479	0.098979	0.120315	15.217095
10	8319583	0.098024	0.098986	0.117233	14.747481
11	6799438	0.098653	0.095579	0.121054	15.004499
12	6541583	0.098987	0.096785	0.122496	15.688028
13	8418758	0.102933	0.103693	0.122805	15.257653
14	8318799	0.103104	0.102834	0.124203	15.534390
15	8228342	0.103145	0.105726	0.124395	15.615664
16	6640758	0.103907	0.112124	0.123357	15.520058
17	8417974	0.105728	0.109075	0.127383	16.017211
18	8129167	0.105947	0.107655	0.131614	16.508599
19	6700263	0.106009	0.113137	0.127541	15.731552
20	6641382	0.106101	0.113261	0.128478	15.818724
21	6541583	0.106539	0.111541	0.123929	16.173154
22	6542207	0.106831	0.111762	0.129386	15.942627
23	6700887	0.107257	0.108942	0.129260	15.821867
24	8417974	0.110204	0.113539	0.132077	16.353993
25	6799438	0.110485	0.114074	0.131099	16.043866
26	6800062	0.111412	0.115665	0.133690	16.219284
27	8318799	0.111725	0.112824	0.136621	17.059014
28	8319583	0.112419	0.115052	0.132746	16.129693
29	8418758	0.113802	0.119962	0.135531	16.390391
30	6640758	0.119583	0.153584	0.137452	17.745088
31	8128383	0.239779	0.490835	0.258188	25.178331

Continued on next page

	n_params	MAE	MRRE	L2E	H1E
32	8227558	0.240262	0.493193	0.258782	25.276054

Table 5.12: *Test 2*: Experiments executed on GINN with 512 training data and 256 validation data.

We would like to again point out to the reader that the GINNs are much lighter networks, as they require a significantly lower number of training parameters compared to the FCNNs. For the first problem, we observe an average of 81371007 parameters for training the FCNNs and an average of 7472196.5 parameters for training the GINNs, i.e., approximately 9% compared to the average for the FCNNs.

5.2.1 Hyperparameter Analysis

Even for this problem, an extensive grid search for hyperparameters has been performed, testing various values for both the FCNNs and the GINNs, as illustrated in Table 4.2. Each hyperparameter have a pair of values to test for each architecture and the goal is to find, in general, the combination of hyperparameters that works best, so each pair of hyperparameters is analyzed individually. The results are presented below.

- *Number of Hidden Layers (nh)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the qualitative results are shown in Figure 5.22. In the case of FCNNs, $nh = 6$ performs slightly better than $nh = 8$, i.e., the opposite with respect the first problem. This result confirm that this parameter is not characterizing. Instead, in the case of GINNs, $nh = 80$ outperforms the other option as for the first problem.

- *Use of Residual connections (rs)*
  This hyperparameter was used exclusively for the FCNNs, and the qualitative results are shown in Figure 5.23. The use of residual blocks appears to worsen the situation.

- *Using Batch Normalization (bn)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the qualitative results are shown in Figure 5.24. In both cases, the use of batch normalization seems to improve the model's performance.

- *Activation Function (af)*
  This hyperparameter is used in both the FCNNs and the GINNs, and the

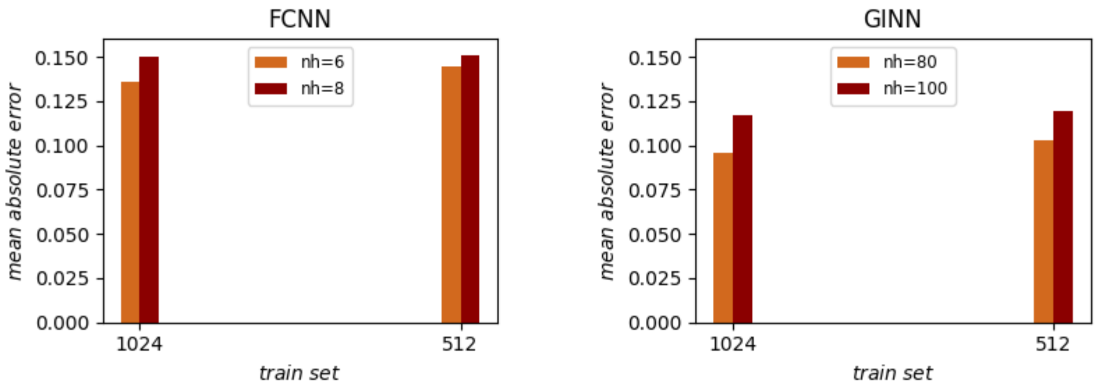


Figure 5.22: Number of Hidden Layers (nh).

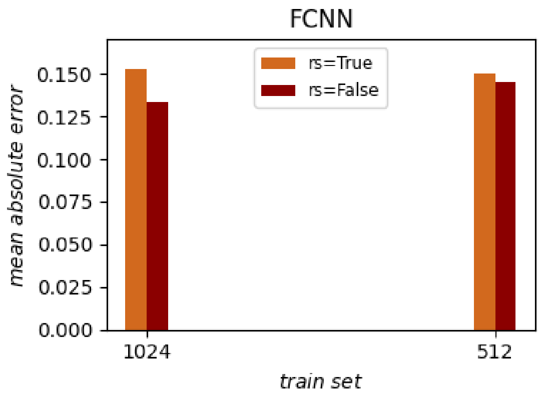


Figure 5.23: Use of Residual connections (rs).

qualitative results are shown in Figure 5.25. In both cases, the difference is clear, for FCNNs, the *mish* activation function performs better, while for GINNs, the *elu* activation function is the one which performs better.

- *Width of Hidden Layers (wh)*
  This hyperparameter was used exclusively for the FCNNs, and the qualitative results are shown in Figure 5.26. As for the first problem, it seems that a layer with a width of 2048 performs better than the other, but the difference is still not significant.

- *Number of Filters (nf)*
  This hyperparameter was used exclusively for the GINNs, and the qualitative results are shown in Figure 5.27. It seems that a layer with 5 filters performs better than the one with 10 but, even in this case, the difference is not

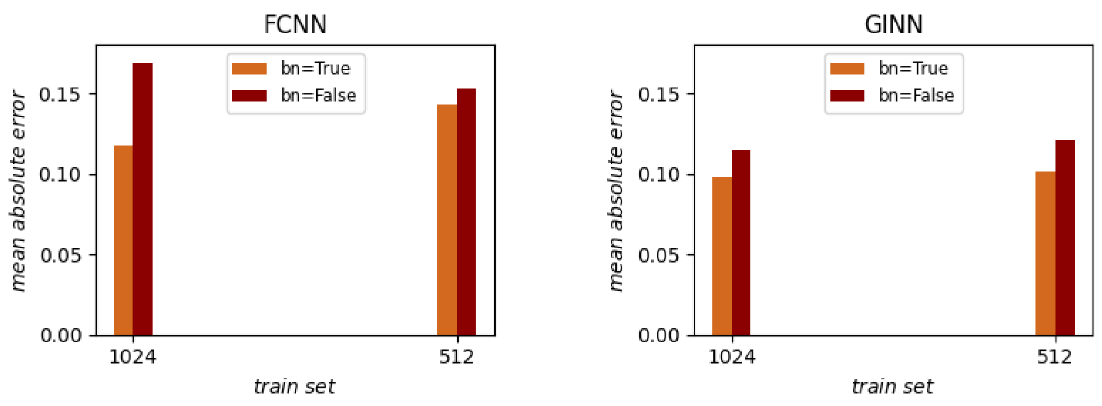


Figure 5.24: Using Batch Normalization (bn).

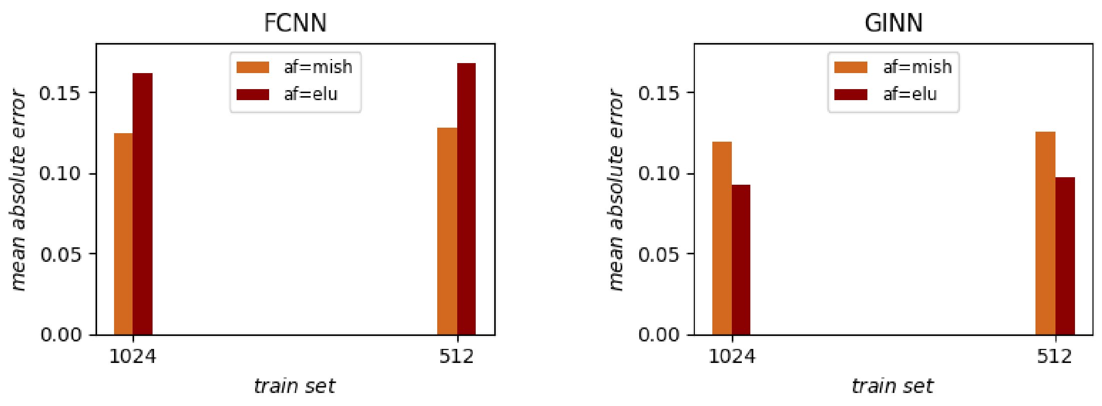


Figure 5.25: Activation Function (af).

significant.

- *Input Refresh (rf)*
  This hyperparameter was used exclusively for the GINNs, and the qualitative results are shown in Figure 5.28. As for the first problem, this hyperparameter proves to be one of the most significant, as the benefit of refreshing the inputs is visually noticeable.

To conclude the analysis of the hyperparameters, we search for the ideal combination for each type of network. The selected combinations are the same as those for the first problem for both the architectures. The observation remains that, the GINNs are more robust in their results, showing a clear combination of ideal hyperparameters, unlike the FCNNs, where no results seem to be always more effective than its counterpart. In Figure 5.29 and Figure 5.30, all the MAEs related to the

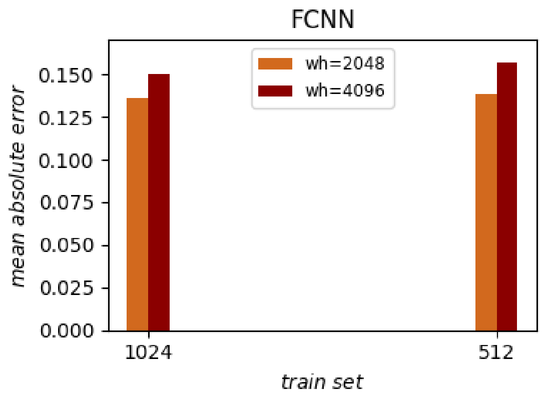


Figure 5.26: Width of Hidden Layers (wh).

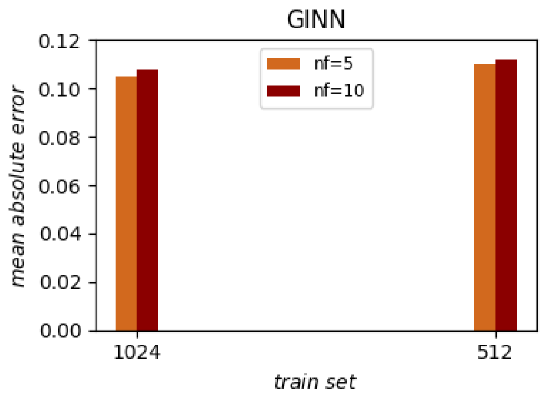


Figure 5.27: Number of Filters (nf).

different configurations for both GINNs and FCNNs are shown, with emphasis on those related to the configurations that contain the hyperparameter combination we identified as the "best". As for the first problem, looking at the figures overall, we can observe that the configurations highlighted as ideal are among the models with the lowest error only for the GINN models and not for FCNN ones. So even in this second problem the FCNNs appear to be less consistent.

5.2.2 Additional Analysis of the Best Model

As shown by the results displayed in Figures 5.1 and 5.21, for *Test 1*, GINNs achieve a very low average error, MAE less than 0.001 and MRRE of about 0.01, while for *Test 2*, despite outperforming the FCNNs, they achieve a MAE of about 0.11 and a MRRE of about 0.12, which are much higher compared to the first

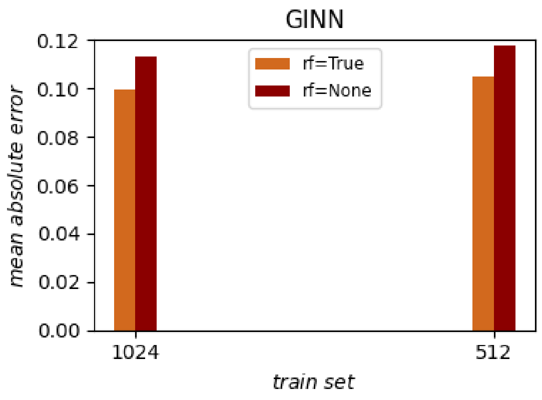


Figure 5.28: Input Refresh (rf).

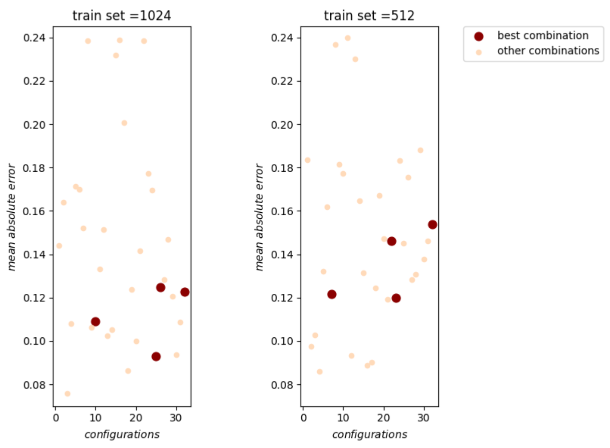


Figure 5.29: Best combination for FCNNs: use of batch normalization = True, width of hidden layers = 2048, activation function = *mish*.

problem. For this reason, we chose to conduct additional analyses regarding the second problem.
We want to analyze whether there are relations between the prediction performances and the parameters, both the physical ones, diffusion and convection, and the geometric ones (BCs). To do this, we select and analyze only one GINN model, the best-performing one according to MAE, which is the model with: number of hidden layers = 80, use of batch normalization = True, number of filters = 5, activation function = *mish* and input's refresh = True. We decided to conduct these additional analyses considering only the MRRE.

The first thing we investigate is whether the physical parameters of the problem somehow influence the predictions. We recall that in this case, the parameter that

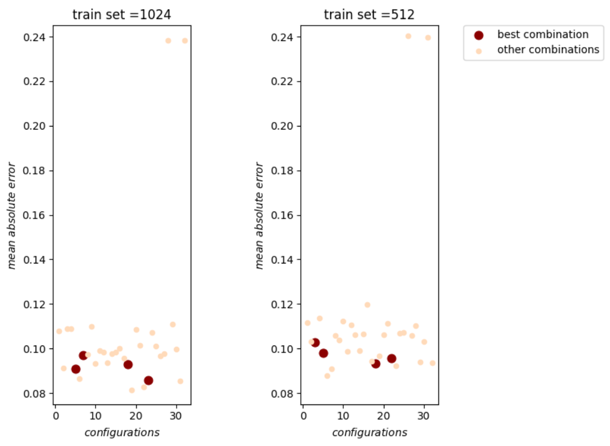


Figure 5.30: Best combination for GINNs: use of batch normalization = True, refresh input = True, activation function = *elu*.

characterizes this specific problem is $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_D]$, where $\mu_1 \in$ (0,5) controls the diffusion intensity and $\mu_2 \in (0, \pi)$ controls the direction of the convection; we call $\mathcal{P}_\mu$ the parametric space defined by (μ_1, μ_2).
From an initial simple analysis, see Figure 5.31, it is noticeable that there are performances visibly worse than the average for fairly identifiable values of μ_1 and μ_2. Specifically, it appears that the performance deteriorates as $\mu_2 \to 0$ and $\mu_1 \to 0$ or $\mu_1 \to \pi$.

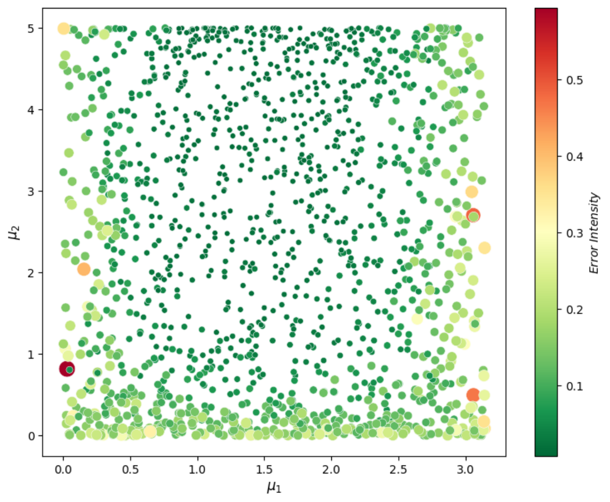


Figure 5.31: MRRE with respect to physical parameters μ_1 and μ_2.

For a more accurate study, we created a function with variables (ϵ_1, ϵ_2) that measures the average prediction error on the test set within the parameter range (μ_1, μ_2) in $[0 + \epsilon_1, \pi - \epsilon_1] \times [\epsilon_2, 5]$, with $\epsilon_1 \in [0, \pi/2]$ and $\epsilon_2 \in [0, 5]$. The 3D plot of the surface created by this function, see Figure 5.32, identifies areas from which it can be observed that there is a region corresponding to a specific interval (ϵ_1, ϵ_2) where the average error is lower.

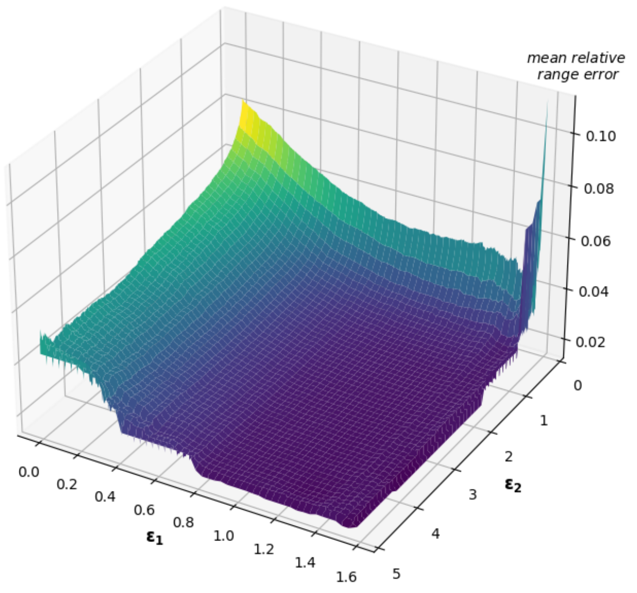


Figure 5.32: MRRE with respect to variables (ϵ_1, ϵ_2).

To represent the variability of the error, we create two additional surfaces: one where we plot the minimum error and the other where we plot the maximum error, both with respect to the variables (ϵ_1, ϵ_2). In Figure 5.33, we see that as we move away from the intervals already identified as the worst, $\mu_2 \to 0$ and $\mu_1 \to 0$ or $\mu_1 \to \pi$, the error range decreases.
To determine specific parameters $(\epsilon_1^*, \epsilon_2^*)$ where the prediction mean error on the test set is the lowest and most consistent, we examine the surface that identifies the maximum error, Figure 5.33. We note that by removing only certain values of the variables (ϵ_1, ϵ_2), it is possible to eliminate the "blocks" of the highest errors. We selected $\epsilon_1^* = 0.34$ and $\epsilon_2^* = 0.28$ and the resulting surfaces are illustrated in Figure 5.34.
With the selected values, we have a new parametric space $\hat{\mathcal{P}}_\mu \subset \mathcal{P}_\mu$, where $\mu_1 \in (\epsilon_1^*, \pi - \epsilon_1^*)$ and $\mu_2 \in (\epsilon_2^*, 5)$. The average error over the entire test set, where the physical parameters belong to $\mathcal{P}_\mu$, is 0.082. However, if we select only the test set

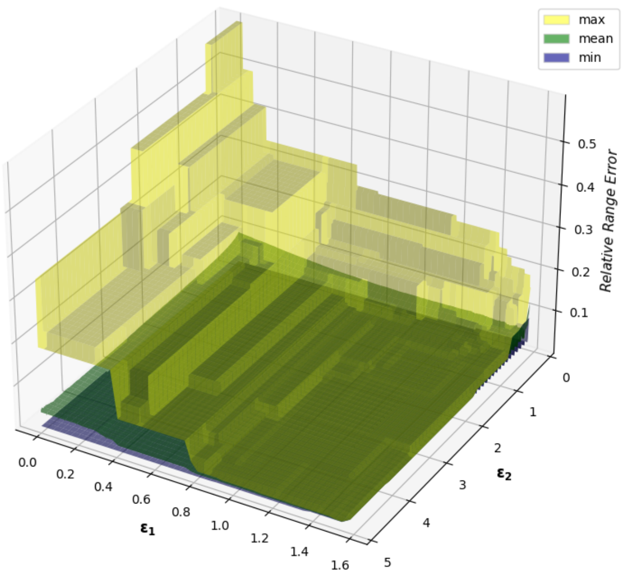


Figure 5.33: Comparison of the minimum, mean and maximum relative range error with respect to variables (ϵ_1, ϵ_2).

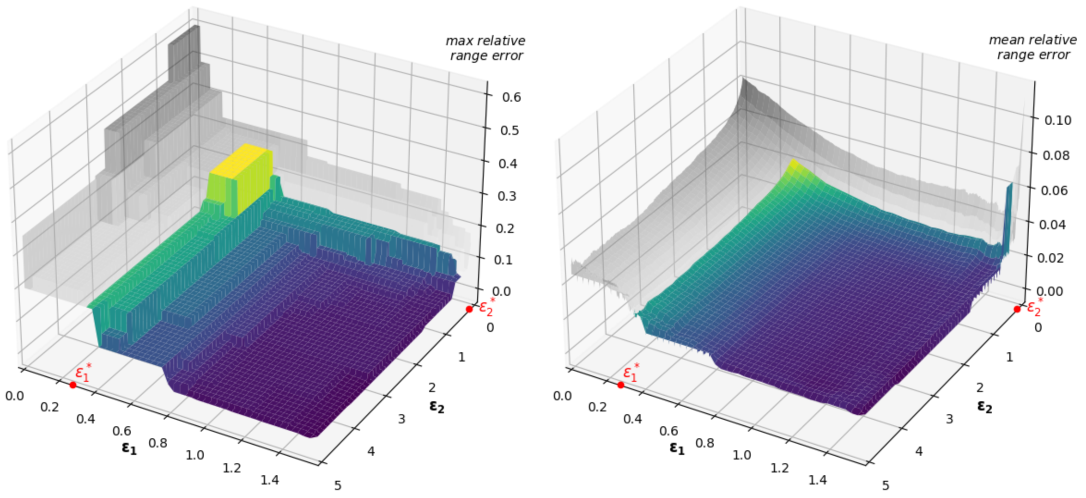


Figure 5.34: Maximum and mean relative range error with respect to the variables (ϵ_1, ϵ_2), with specific threshold values $(\epsilon_1^*, \epsilon_2^*)$.

where the physical parameters belong to $\hat{\mathcal{P}}_\mu$, we obtain an average error of just 0.047. Thus, by reducing the parametric space by approximately 13.6% of the values, we decrease the average MRRE by 42.4%.

The last analysis regard the influence of the geometric parameters, i.e., the BCs. To see if there is any correlation between the different configurations of BCs and the predictions made by the model, we plot the BCs corresponding to the best and worst nine predictions, see Figures 5.35 and 5.36. In these figures, we have also included the physical parameters to seek further correlations.

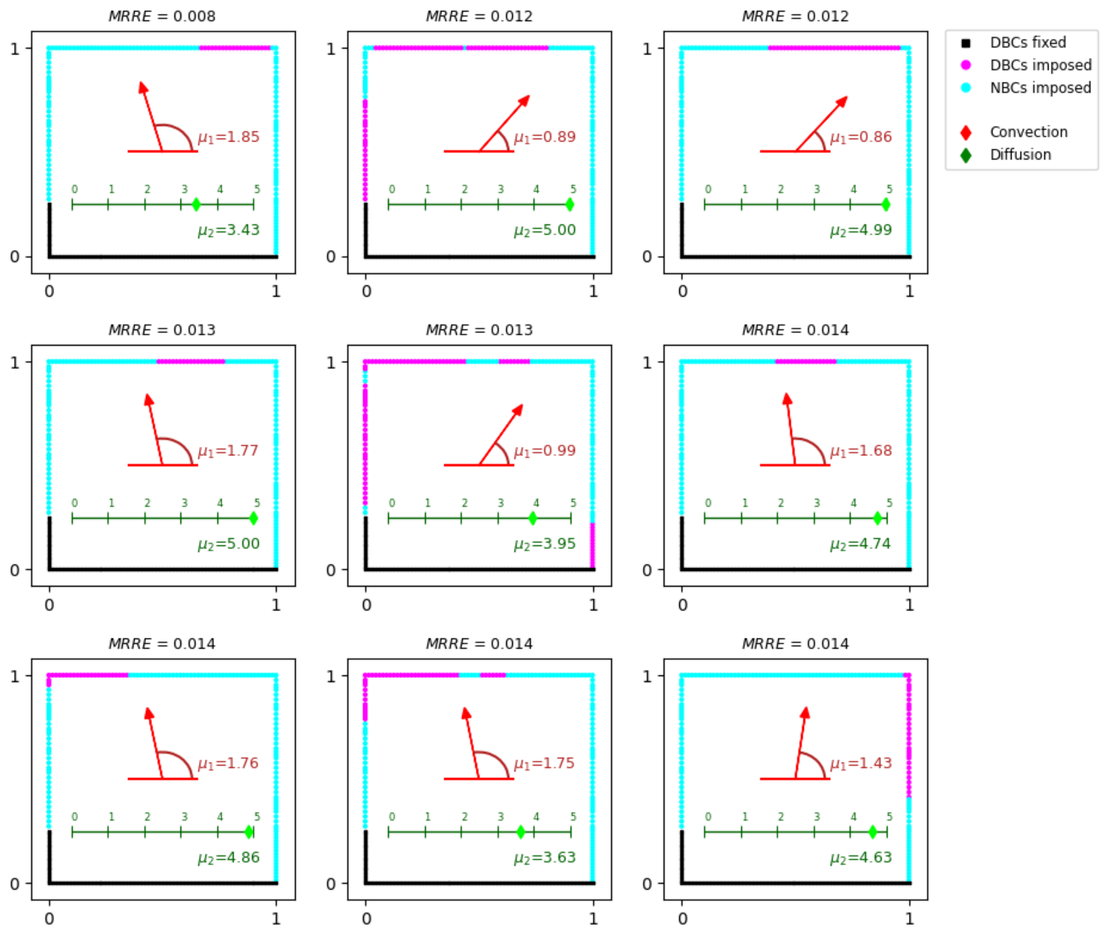


Figure 5.35: Geometrical and physical parameters related to the best predictions.

A very first observation we can draw from these graphs is that the parameters (μ_1, μ_2) of all the cases concerning the best predictions belong to the parametric space $\hat{\mathcal{P}}_\mu$, while the parameters of all the cases describing the worst predictions belong to the parametric space $\mathcal{P}_\mu \setminus \hat{\mathcal{P}}_\mu$.
For the sake of knowledge, we report in Figure 5.37 the worst predictions, considering only the test cases having $(\mu_1, \mu_2) \in \hat{\mathcal{P}}_\mu$. Comparing Figures 5.35 and 5.37,

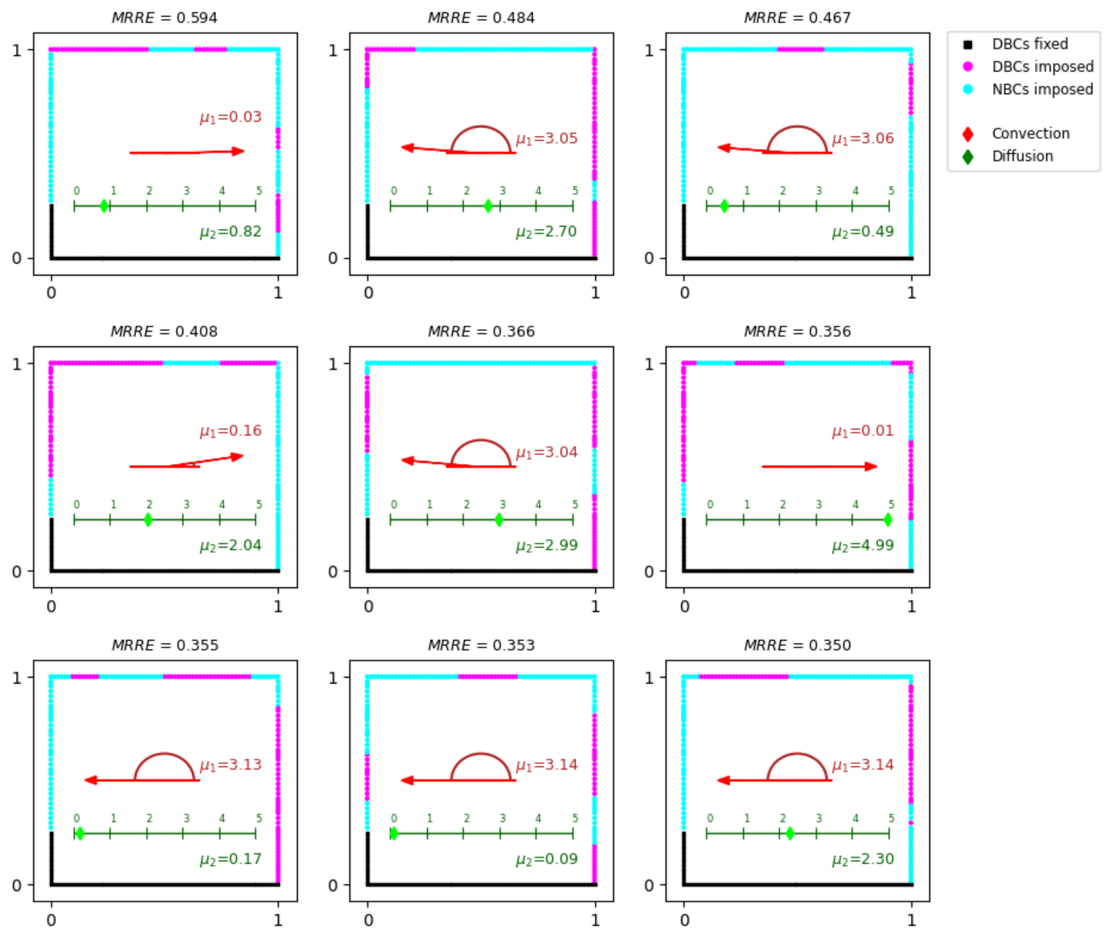


Figure 5.36: Geometrical and physical parameters related to the worst predictions.

we can focus solely on the BCs and no longer on the physical parameters, as we have already eliminated the extreme cases. However, it is not possible to identify a clear geometrical pattern within this data.

5.2.3 Qualitative Analysis of the Best Model

About *Test 2*, the best-performing GINN configuration according to MAE is the one with: number of hidden layers = 80, use of batch normalization = True, number of filters = 5, activation function = *mish* and input's refresh = True. We note that the best configuration for *Test 2* is very similar to that found for *Test 1*; only one hyperparameter, the activation function, differs.
Let us conclude the section with a qualitative analysis by showing some cases from this configuration, comparing our model's predictions with the ground truth

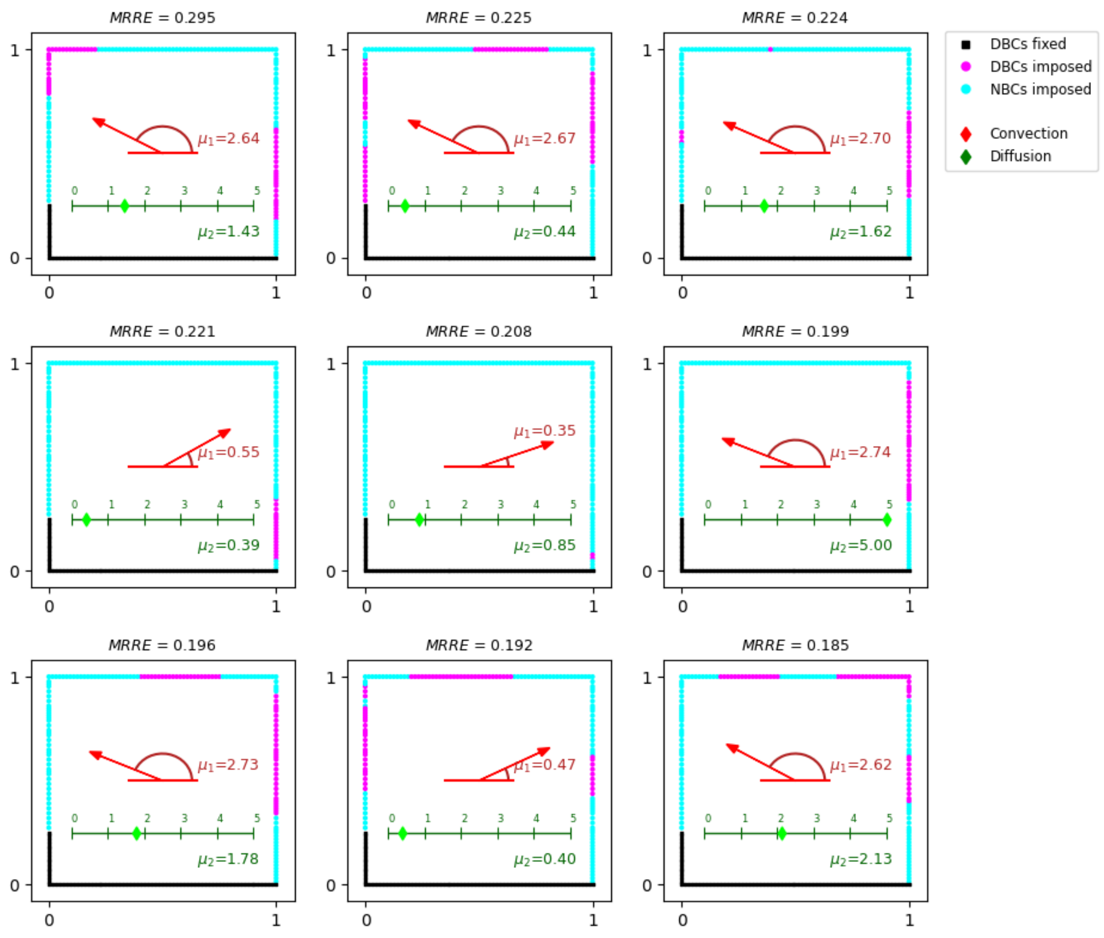


Figure 5.37: Geometrical and physical parameters related to the worst predictions, considering only the test cases having $(\mu_1, \mu_2) \in \hat{\mathcal{P}}_\mu$.

solutions. In particular, the two best cases, Figures 5.38 and 5.39, and two worst cases, Figures 5.44 and 5.45, with respect to MAE and the two best cases, Figures 5.40 and 5.41, and two worst cases, Figures 5.46 and 5.47, with respect to MRRE. In addition in Figures 5.42 and 5.43 we report the predictions from two random test cases.

By observing the figures, we can make some considerations. First of all, regarding the MAE, we can observe that the best predictions are those where the square boundary is almost entirely under a Neumann BC of zero value, except for a single window on the top side where Dirichlet condition is imposed to 0. In general, both for MRRE and MAE, we see the best predictions among the problems where the angle identifying the direction of the convection vector is close to $\pi/2$, i.e., pointing towards the top side of the square ($\mu_1 \to \pi/2$), and the value of μ_2 is quite high (at least $\mu_2 > 2.5$). Among the worst predictions, we highlight as a common feature the convection vector being directed almost horizontally, i.e., pointing towards the left or right side of the square ($\mu_1 \to 0$ or $\mu_1 \to \pi$).

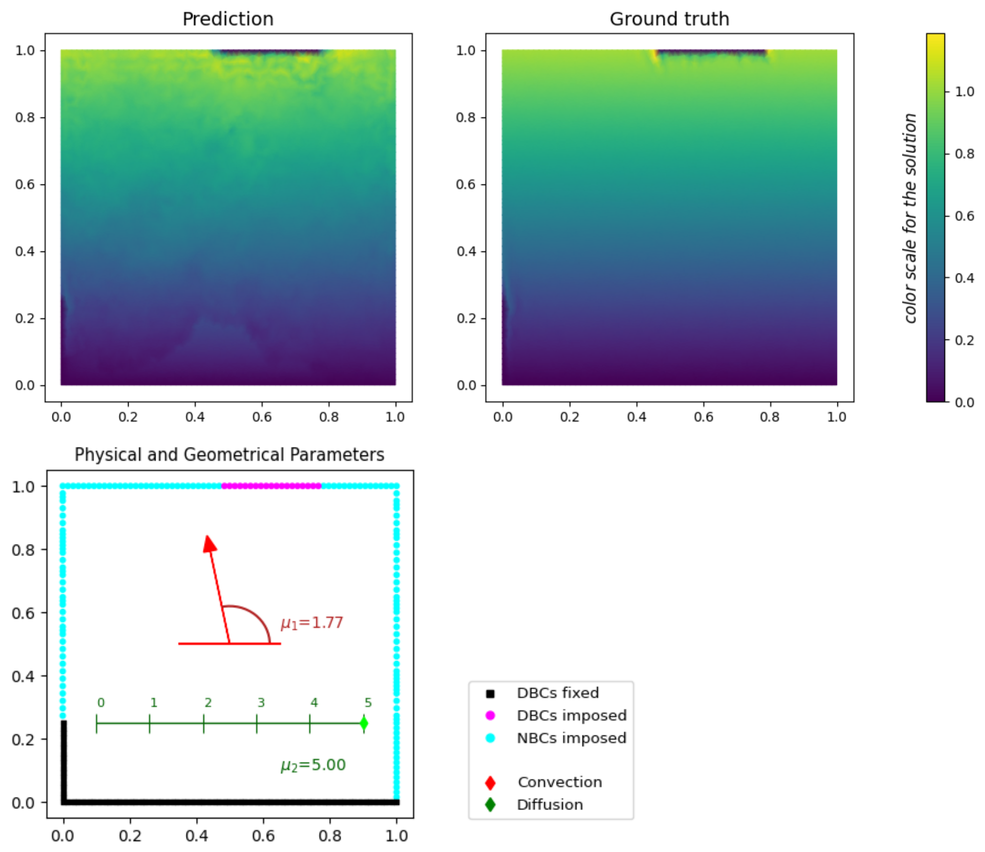


Figure 5.38: Best prediction with respect to MAE, $MAE = 5.6068e^{-05}$ and $MRRE = 0.01223$.

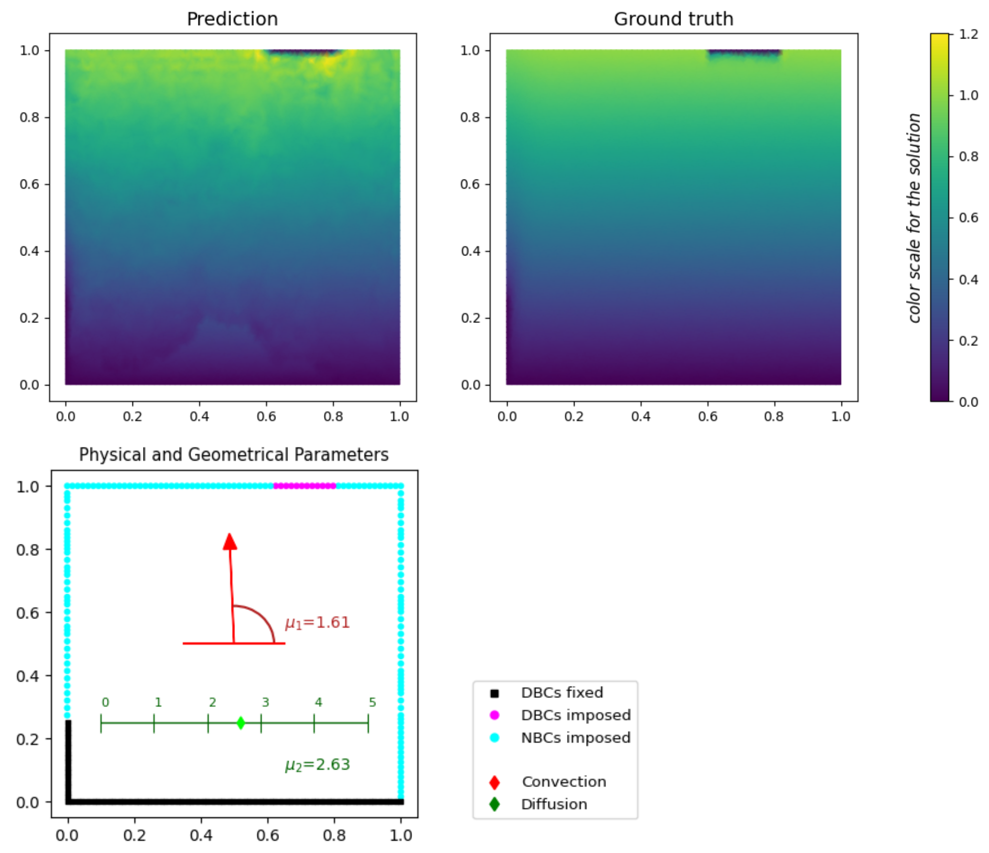


Figure 5.39: Second best prediction with respect to MAE, $MAE = 5.6155e^{-05}$ and $MRRE = 0.00592$.

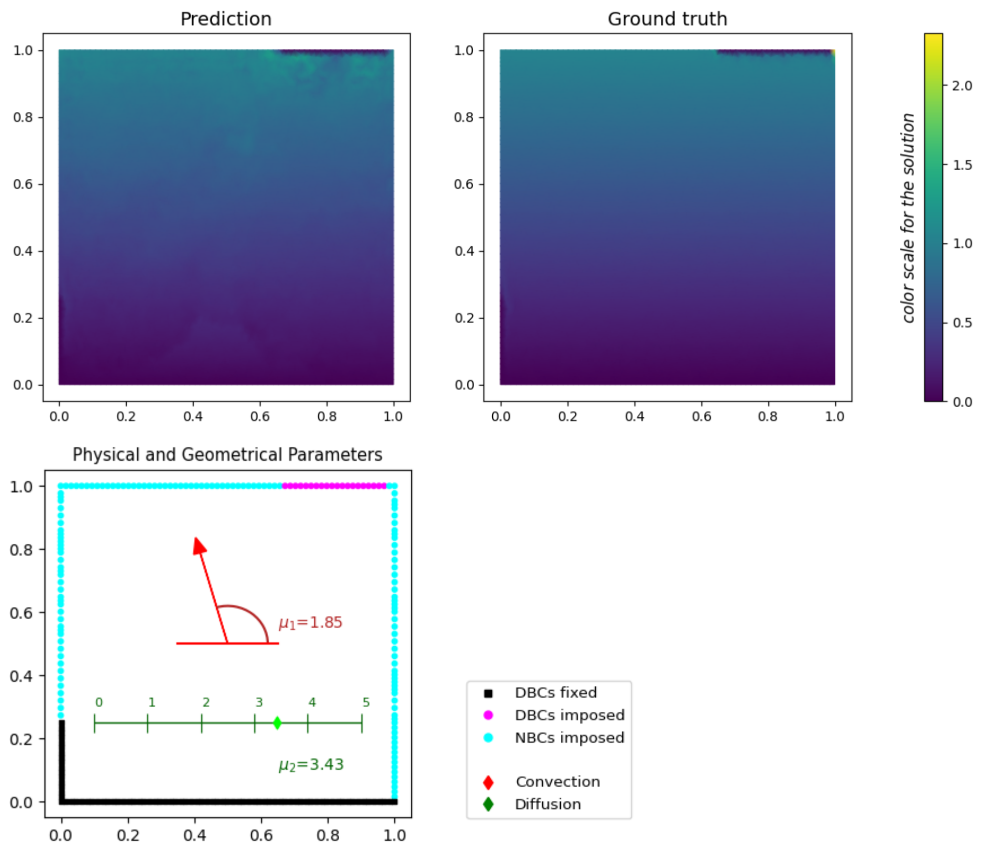


Figure 5.40: Best prediction with respect to MRRE, $MAE = 0.000168$ and $MRRE = 0.00105$.

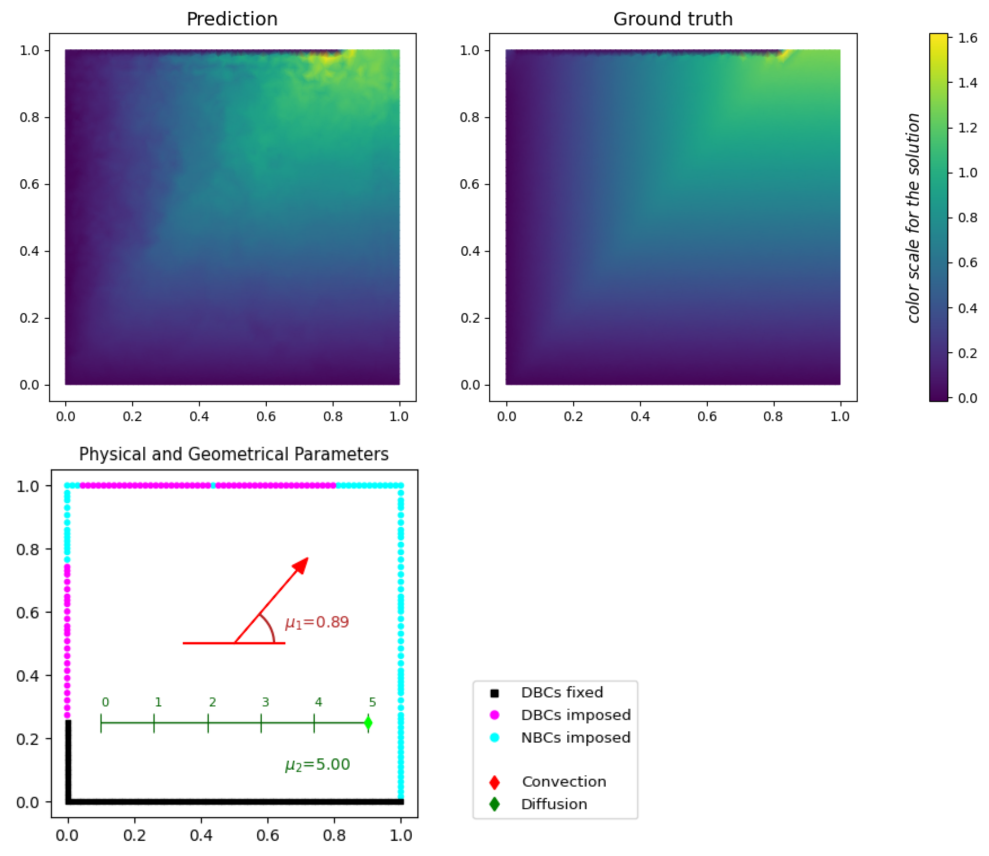


Figure 5.41: Second best prediction with respect to MRRE, $MAE = 0.00021$ and $MRRE = 0.00107$.

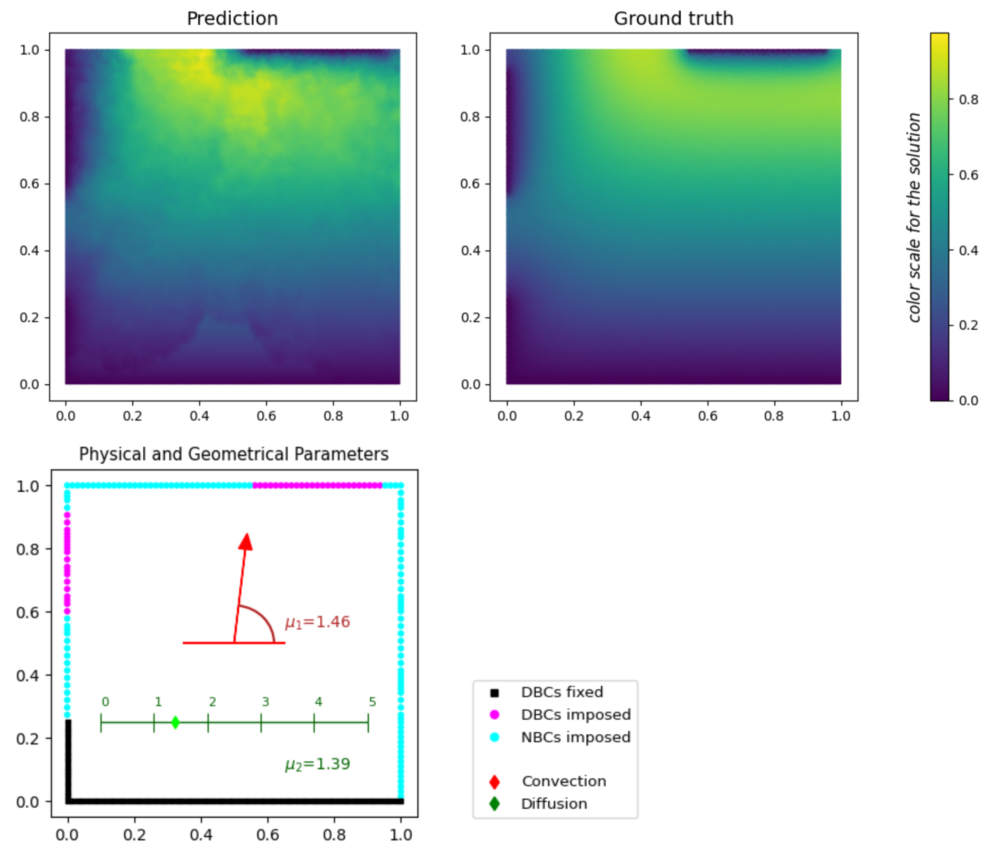


Figure 5.42: A random prediction, $MAE = 0.00028$ and $MRRE = 0.00178$.

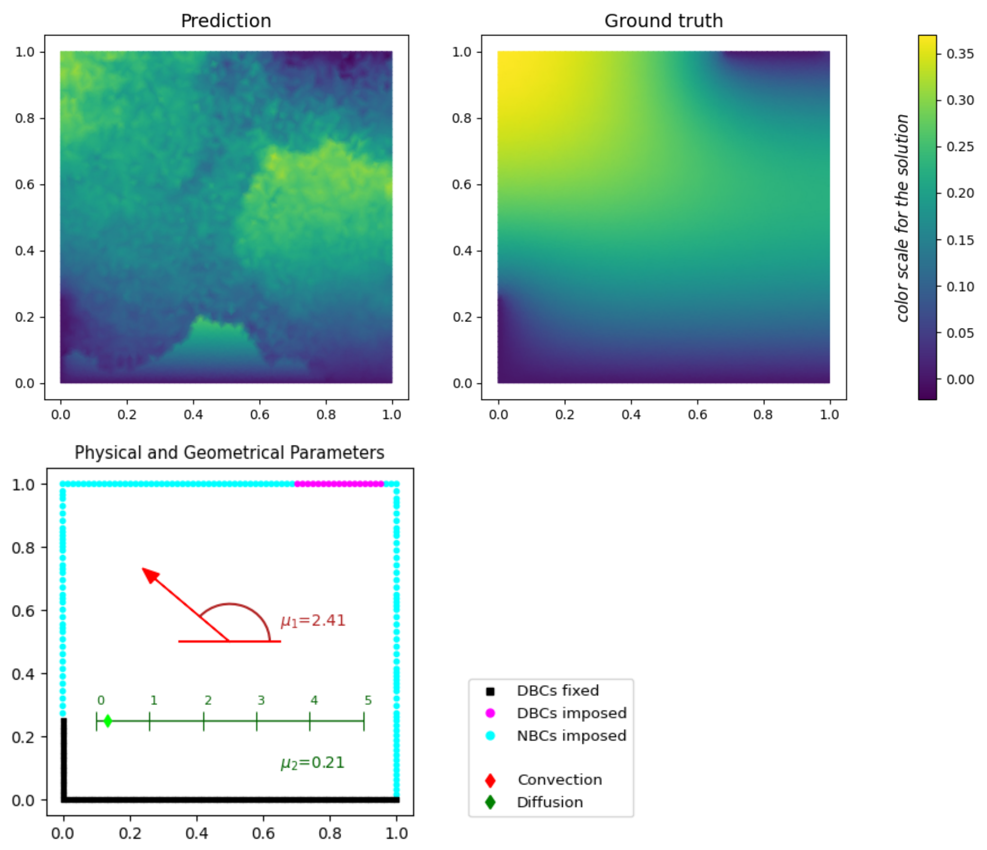


Figure 5.43: A second random prediction, $MAE = 0.00041$ and $MRRE = 0.01080$.

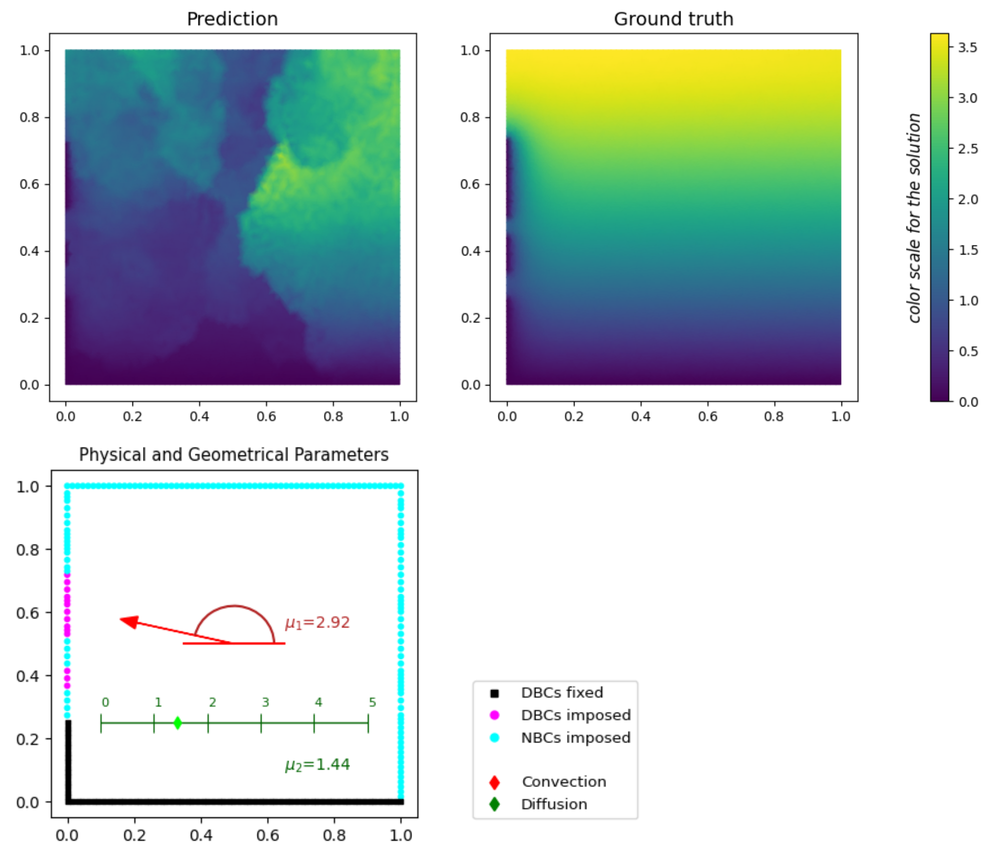


Figure 5.44: Worst prediction with respect to MAE, $MAE = 0.00300$ and $MRRE = 0.01133$.

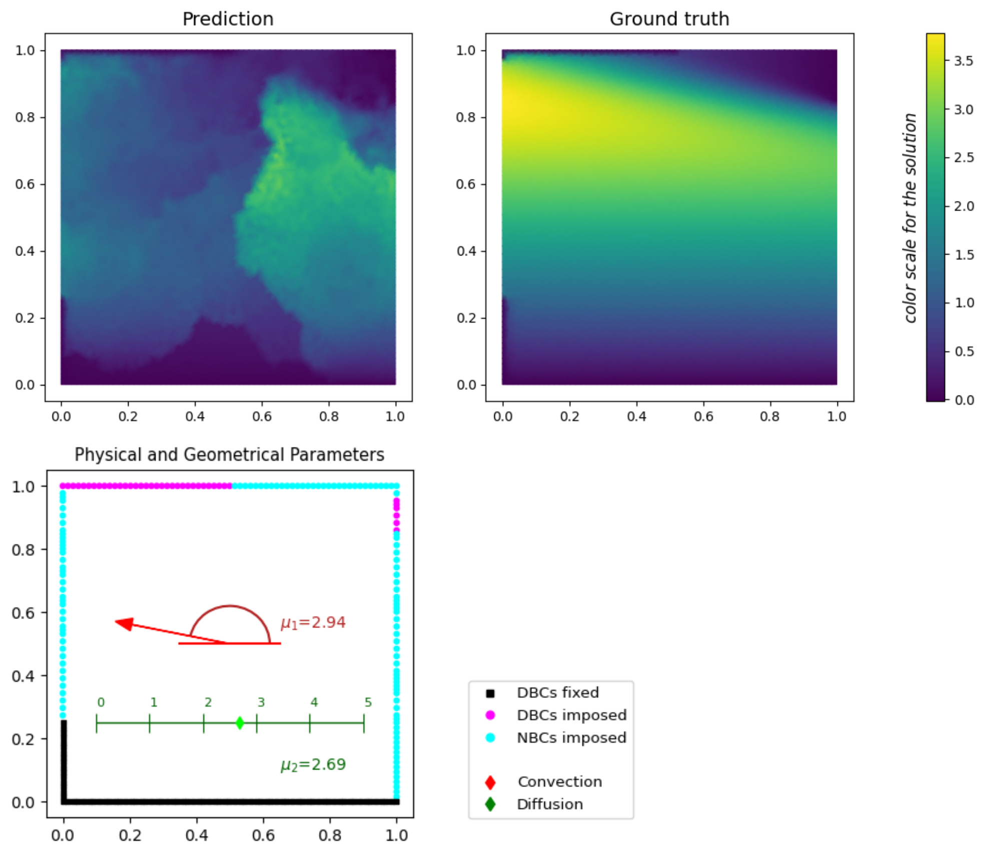


Figure 5.45: Second worst prediction with respect to MAE, $MAE = 0.00268$ and $MRRE = 0.00942$.

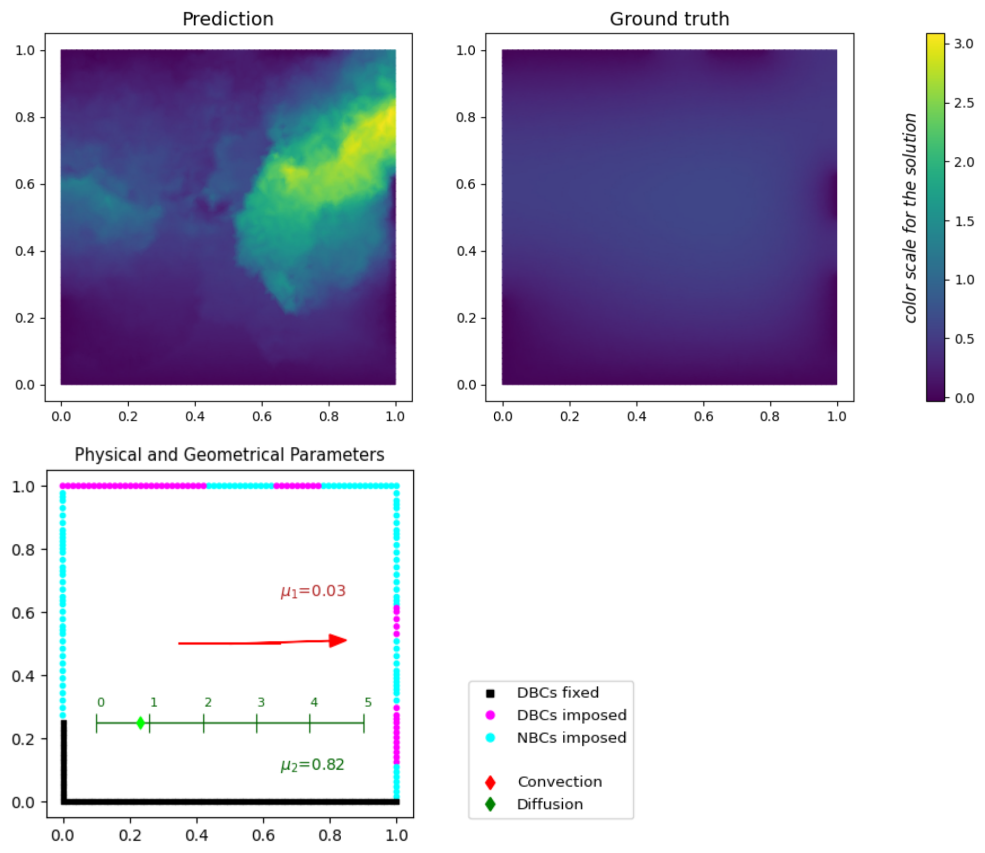


Figure 5.46: Worst prediction with respect to MRRE, $MAE = 0.00014$ and $MRRE = 5.07227$.

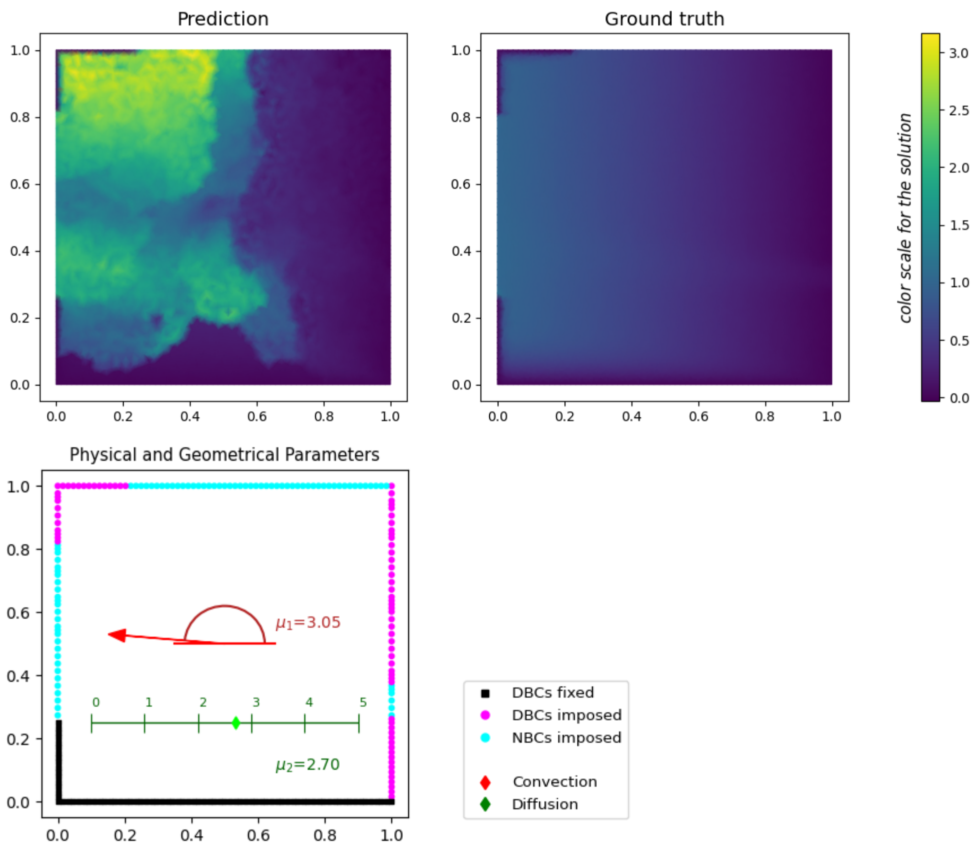


Figure 5.47: Second worst prediction with respect to MRRE, $MAE = 0.00014$ and $MRRE = 1.39954$.

6 Conclusions

This is the concluding chapter, where we present an overview of the results of the entire thesis project and offer some ideas for future research and applications.
The focus of this work is on training surrogate Deep Learning (DL) models to predict solutions to parametric Partial Differential Equation (PDE) problems with a specific focus on parametrized boundary conditions (BCs). The test cases we have prepared are based on graph-structured data originated from the mesh of the numerical solver used for simulating the ground truth data. What makes this approach innovative is the fact that most the parameters are based on the nodes of the graph structure to describe the parametrized BCs, not just to describe the physical properties.
To better exploit the graph structure of our data, we used the Graph-Informed Neural Network (GINN), a new architecture that extends the basic formulation of spatial-based Graph Convolutional Networks (GCN). This architecture was specifically designed for regression tasks like ours, and furthermore, the distinguishing layer (the GI layer) leverages the adjacency matrix of the given graph to determine the unit connections within the network.
As discussed several times, one of the main issues with standard GCNs is that, in many cases, they fail to outperform even a simple Fully Connected Neural Network (FCNN), and one of the key goals of GINNs, is to show that they can indeed exceed the performance of traditional Neural Networks (NNs). The most significant result we found is the confirmation that GINNs outperform FCNNs; in both test cases we have analyzed, the mean errors for the GINNs is consistently lower than that of the FCNNs.

The GINNs have demonstrated many qualities, including robustness, reliability, and efficiency. To proceed in order, we observed that GINNs appear to be robust, as in both problems, there is no significant difference in performance between simulations with 1,024 training data points and those with 512 (half the amount). Furthermore, the reliability of GINNs is proven by the fact that they have an ideal configuration of hyperparameters, which is consistently more effective than its counterparts. In contrast, FCNNs lack an optimal configuration, making their results more random compared to those from GINNs. Finally, GINNs also have the significant advantage of a smaller dimension with far fewer weights compared to FCNNs, making them efficient models that require less memory without sacrificing approximation quality.

Going into more detail, the first problem addressed is a classic PDEs problem of diffusion, and GINNs demonstrate their ability to handle it remarkably well, as the errors obtained are very low. For this reason, we decided to test the new network on a much more complex problem that involves two physical phenomena, diffusion and convection. In addition to the geometric parameters related to the BCs, there are also physical parameters, which we found to significantly affect the model performance.
For this second test case, we conducted additional analyses, particularly to guide future developments related to this specific problem. We discovered a range of physical parameters where the network's performance is significantly lower than average. Fortunately, this range of parameters turned out to be quite clear and defined, making it easier to address and eliminate them. This allows us to conduct further analyses without the results being influenced by the physical parameters already identified as "detrimental", achieving lower average errors for this problem as well.
Among these analyses, we focused on studying the BCs. We examined how the geometric parameters of the problem might influence the network's predictions, but we did not find a clear pattern. This suggests that the field (the quantity we want to predict) has very complex characteristics when there is a transition between Neumann and Dirichlet BCs. So, for this problem, we can conclude that the solution is mainly influenced by regime changes, diffusion-convection, and by the extreme angles of the direction of the convection vector; while it seems that it is not evidently influenced by the geometry of the BCs.

In conclusion, this thesis demonstrates that GINNs are a promising surrogate model for efficiently solving parametric PDEs with low computational costs. The GINN, with its graph-based design, allows for the efficient handling of complex parametric spaces, offering more accurate predictions than those achieved with traditional DL models.

Future Works

In this thesis project, we tackled both a simple and a highly complex case of PDEs, demonstrating that GINNs can adapt to different types of problems. In the future, additional cases of varying difficulty and complexity could be explored, for example, Navier-Stokes problem.
Moreover, further variability in the geometry could be explored, such as introducing a hole that moves or changes in size, in *Test 1* or introducing some parts of the boundary where the BCs are non-homogeneous in *Test 2*. This added complexity would allow us to assess the robustness and adaptability of the model when faced with dynamic geometric features. By testing how the network responds to these

variations, we could better understand its potential for handling more complex and realistic physical scenarios. This would also offer valuable insights into how well GINNs generalize across different geometric configurations.
Another one promising direction for this work lies in the field of Optimal Control applications. NNs, due to their speed and differentiability, can be leveraged to determine the parameters, both physical and geometrical (BCs), of a problem that generate solutions satisfying specific desired characteristics. By using the efficiency of NNs to quickly explore a wide range of parameter spaces, we can optimize control strategies and identify configurations that yield the most suitable or optimal solutions for complex systems governed by PDEs. This approach could be especially valuable in common scenarios where traditional methods are computationally expensive or impractical.

Bibliography

[1] Afshine Amidi and Shervine Amidi. Convolutional Neural Networks cheatsheet. *Stanford Univeristy*, 2019.

[2] Benjamin Anderson. Distribuzione di Probabilità. *Statorials*, 2023.

[3] P. Benner, S. Grivet Talocia, A. Quarteroni, G. Rozza, W. Schilders, and L.M. Silveira. *Model Order Reduction.* De Gruyter, 2020.

[4] S. Berrone, F. Della Santa, A. Mastropietro, S. Pieraccini, and F. Vaccarino. Layer-wise relevance propagation for backbone identification in discrete fracture networks. *Journal of Computational Science*, 2021.

[5] S. Berrone, F. Della Santa, A. Mastropietro, S. Pieraccini, and F. Vaccarino. Graph-Informed Neural Networks for Regressions on Graph-Structured Data. In *Mathematics*, 2022.

[6] Manomita Chakraborty. Explainable Neural Networks: Achieving Interpretability in Neural Models. *Archives of Computational Methods in Engineering*, 2024.

[7] Shing Chan and Ahmed H.Elsheikh. A machine learning approach for efficient uncertainty quantification using multiscale methods. *Journal of Computational Physics*, 2018.

[8] D. Coscia, L. Meneghetti, N. Demo, G. Stabile, and G. Rozza. A continuous convolutional trainable filter for modelling unstructured data. *arXiv, Computer Science*, 2022.

[9] Francesco Della Santa. Sparse Implementation of Versatile Graph-Informed Layers. *ArXiv, Computer Science*, 2024.

[10] K. Doherty, C. Simpson, S. Becker, and A. Doostan. QuadConv: Quadrature-Based Convolutions with Applications to Non-Uniform PDE Data Compression. *arXiv, Computer Science*, 2023.

[11] N.R. Franco, A. Manzoni, and P. Zunino. Learning Operators with Mesh-Informed Neural Networks. *arXiv, Computer Science*, 2022.

[12] S. Fresca, L. Dede, and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Computational Science*, 2021.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[14] I.Goodfellow, Y.Bengio, and A.Courville. Deep Learning. *MIT Press*, 2016.

[15] Atwood J. and Towsley D. Diffusion-Convolutional Neural Networks. *In Advances in Neural Information Processing Systems; Curran Associates*, 29, 2016.

[16] Zichao Jiang, Junyang Jiang, Qinghe Yao, and Gengchao Yang. A neural network-based PDE solving algorithm with high precision. *Scientific Reports*, 2023.

[17] Steven G. Johnson. Notes on Separation of Variables. *MIT OpenCourseWare*, 2012.

[18] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with Artificial Neural Networks. *European Journal of Applied Mathematics*, 2017.

[19] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 2021.

[20] I. E. Lagaris, A. Likas, and Fotiadis. Artifcial neural networks for solving ordinary and partial diferential equations. *IEEE Transactions on Neural Networks*, 1998.

[21] K. Lee and K.T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 2020.

[22] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *ArXiv, Computer Science*, 2018.

[23] Wei Ma and Jun Lu. An Equivalence of Fully Connected Layer and Convolutional layer. *Technical Report*, 2017.

[24] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics & Fluids*, 2021.

[25] A. Micheli. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Transactions on Neural Networks*, 20, 2009.

[26] M. Morimoto, K. Fukami, K. Zhang, A.G. Nair, and K. Fukagata. Convolutional neural networks for fluid flow analysis: toward effective metamodeling and low dimensionalization. *Theoretical and Computational Fluid Dynamics*, 2021.

[27] F. Pichi, F. Ballarin, G. Rozza, and J.S. Hesthaven. An artificial neural network approach to bifurcating phenomena in computational fluid dynamics. *Computers & Fluids*, 2023.

[28] Federico Pichi, Beatriz Moya, and Jan S.Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. *Journal of Computational Physics*, 2024.

[29] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 1999.

[30] Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 2019.

[31] Alfio Quarteroni. *Numerical Methods for Differential Problems.* Springer, 2012.

[32] Maziar Raissi and George Em.Karniadakis. Machine Learning of Linear Differential Equations using Gaussian Processes. *Journal of Computational Physics*, 2017.

[33] Deep Ray and Jan S. Hesthaven. An artificial neural network as a troubled-cell indicator. *Journal of Computational Physics*, 2018.

[34] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[35] Francesco Della Santa. *Data-Driven Deep Learning Methods for Physically-Based Simulations.* PhD thesis, Univesità degli Studi di Torino, 2021.

[36] A. Sharma, E. Vans, D. Shigemizu, K.A. Boroevich, and T. Tsunoda. Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific Reports*, 2019.

[37] Jan S.Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations.* Springer International Publishing, 2016.

[38] Jared Speck. Introduction to PDEs. *MIT OpenCourseWare*, 2011.

[39] Wikipedia the Free Encyclopedia. Residual Neural Network, 2024.

[40] Wei Wang and Qing Li. Universal Approximation Theory: The Basic Theory for Deep Learning-Based Computer Vision Models. *eprint arXiv, Computer Science*, 2024.

[41] Yating Wang, Siu Wun Cheung, Eric T. Chung, Yalchin Efendiev, and Min Wang. Deep multiscale model learning. *Journal of Computational Physics*, 2020.

[42] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record 4*, 1960.

[43] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32, 2021.

[44] Xiangyun Xiao, Yanqing Zhou, Hui Wang, and Xubo Yang. A Novel CNN-Based Poisson Solver for Fluid Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 26, 2020.

[45] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 2019.