Modeling two-phase flow in porous media using physics-informed neural networks for applications in liquid composite molding

John Hanna

COLLEGE SCIENCES DOCTORAL DE L'INGENIERIE PAYS DE LA LOIRE ET DES SYSTEMES

CENTRALE NANTES

MEMOIRE DE DOCTORAT DE

L'ECOLE CENTRALE DE NANTES

ECOLE DOCTORALE N° 602
Sciences de l'Ingénierie et des Systèmes
Spécialité: Mécanique des solides, des matériaux, des structures et des surfaces

Par

John HANNA

Modeling two-phase flow in porous media using physics-informed neural networks for applications in liquid composite molding

Projet de recherche doctoral présenté et soutenu à Nantes, le 11 Décembre 2023
Unité de recherche : UMR 6183, Institut de Recherche en Génie Civil et Mécanique (GeM)

Rapporteurs avant soutenance :

Chung Hae PARK	Professeur de l'École des Mines	IMT Nord Europe
Emmanuel BARANGER	Directeur de recherche CNRS	ENS Paris-Saclay

Composition du Jury :

Présidente:	Diana MATEUS	Professeure des universités	Ecole Centrale de Nantes
Examinateurs:			
	Nahiene HAMILA	Professeur des universités	ENI Brest
	Chung Hae PARK	Professeur de l'École des Mines	IMT Nord Europe
	Emmanuel BARANGER	Directeur de recherche CNRS	ENS Paris-Saclay
Directeur de recherches doctorales:	Sébastien COMAS-CARDONA	Professeur des universités	Ecole Centrale de Nantes
Co-enc.de recherches doctorales :	Domenico BORZACCHIELLO	Maître de conférences	Ecole Centrale de Nantes

Invité

Yves LE GUENNEC	Dr, Ingénieur R&D,	IRT Jules Verne, Bouguenais

Acknowledgements

I extend my deepest gratitude to the remarkable individuals who played an indispensable role in the completion of this work. Their unwavering support and invaluable contributions have made this achievement possible.

I express my sincere thanks to the esteemed members of the jury for their insightful discussions and constructive questions, which have greatly enriched my understanding of the subject and illuminated promising avenues for future research. A special acknowledgment goes to the thesis reporters for their dedicated time and effort in thoroughly reviewing my manuscript. Their constructive and inclusive feedback not only underscored the strengths of the thesis but also provided valuable suggestions for further enhancement.

The work couldn't have been achieved without the help and support of my esteemed supervisors, Sebastien and Domenico, to whom I owe profound respect and gratitude. Sebastien's guidance throughout the entire thesis process, particularly in navigating the intricacies of composite manufacturing, has been instrumental. Domenico, it has been a pleasure working with you; the fruitful outcomes of our joint efforts are evident in this thesis. For both of you, although our formal collaboration may conclude, I still look forward to the prospect of continued cooperation and who knows maybe we can share future projects.

Heartfelt thanks are extended to my family, to whom I dedicate this work. Their unwavering support, even across different continents, has been the bedrock of my perseverance. I am immensely grateful for their continuous encouragement. I would also like to express my gratitude to my friends and my loved one, whose presence has made my life away from home warm and comforting. Their companionship has transformed living abroad into a welcoming haven, and for that, I am truly thankful.

Abstract

Liquid composite molding (LCM) is a popular family of composite manufacturing processes, in which a liquid resin is injected into a mold where a textile is set there. The process involves flow in fibrous porous media. Variabilities within and between textile samples can arise due to fabric intrinsic geometrical defects, mishandling, misalignment, and other factors. These inconsistencies can result in marked deviations between the actual and anticipated filling patterns, leading to variations in the manufactured parts quality. This thesis has two main objectives. The first is to build an online framework that predicts the possibility of defects, which helps taking corrective decisions. The second is to improve the characterization of key material properties before the injection starts. To achieve these tasks, we use physics-informed neural networks (PINN). PINN is a technique that is based on merging the data knowledge along with the knowledge of physics, represented by partial differential equations. To target the first objective, PINN is used to build metamodels of the process with parameters of interest as the permeability or inlet boundary conditions. These models are trained offline and can be quickly employed for online predictions. Towards the second objective, a self-supervised learning framework was built based on PINN and convolutional neural networks to identify the permeability tensor field from 2D textile images. The framework shows promising results through comparing with existing experimental flow front images.

Contents

Introduction

Composite materials are formed by combining two or more materials that have different properties to gain from the benefits of both. An example of that is fiber-reinforced composites; they are materials formed by mixing a polymeric resin with fibers such as glass, flax, or carbon fibers. The resulting materials will have good strength due to the fibers while being lightweight due to the polymeric matrix constituent. Due to their enhanced properties, composites are used in the manufacturing of many parts of planes, boats, and cars saving a considerable amount of energy.

During the manufacturing of composites, defects can take place leading in certain cases to part rejection. The reason for the existence of these defects is variabilities which makes the process less robust and predictable. The variabilities can be divided into two categories: process variability and material variability. The process variability has to do with the working conditions of the process such as the working temperature or pressure which might change from one part to another one. Material variability occurs due to variability of material properties within the part or from a part to another, for instance, variability in the areal weight or fiber orientations.

One way to avoid the creation of defects is to have online control over the process. This can be done by having sensors in specific locations during the process. These sensors send information to a controlling agent that assesses the data and makes predictions in real time. According to these predictions, decisions can be made for example to change the pressure or working temperature. In this work, Physics-Informed Neural Networks (PINN) is used as the controlling agent, which is a machine learning technique that can be used in low-data regimes since the known physics is used along with the few data provided by the sensors. Figure 1 is a diagram showing the process of online control using PINN as a controlling agent.

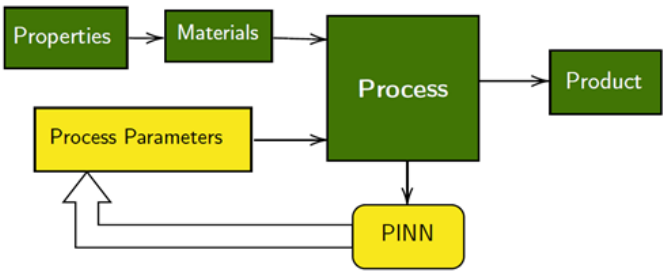

Figure 1: Diagram of a process where sensor data is sent to the controlling agent PINN taking decisions in real-time to change the process parameters accordingly.

Another way to avoid defects is to have proper material characterization specifically tailored to the part in hand. This can improve the predictions through accurate simulations before the process. These predictions can be used to make modifications before the process to avoid defects. Accurate material characterization can be achieved through performing several experiments (process) where different measurable changes are made in each experiment, an example of the changes can be fiber orientation. Material properties are identified for each experiment using PINN as shown in figure 2. An empirical law can be drawn from the collected data between the changed variable and the identified properties. This law can then be used for future material characterization to perform simulations before the process and tailor a suitable process command.

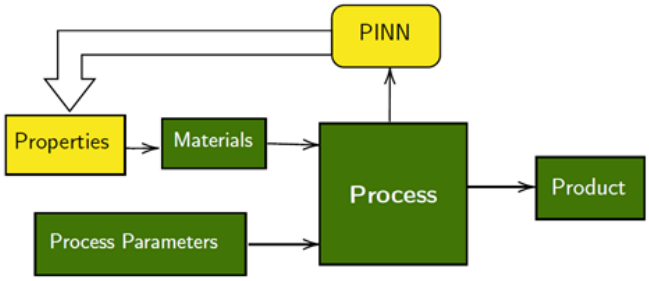

Figure 2: Diagram of a using PINN for proper material characterization where data is transferred from sensors to PINN which then identifies the unknown material properties. In this case, the process parameters are fixed and known.

In this work, techniques are developed based on PINN and deep learning to improve a common family of composite manufacturing processes called liquid composite molding (LCM). The objectives of this thesis can be summarized in two main points which are

1. Improve LCM process robustness through online control,
2. Improve material characterization needed for accurate simulations before the process.

The thesis is organized in the following manner. Chapter 1 introduces the state of the art of deep learning and PINN along with recent machine learning applications in composites manufacturing. Chapter 2 deals with the use of PINN as a solver or simulation tool for two-phase flow in porous media. Chapter 3 details the development of surrogate models using PINN, which is a first step towards online control since such models can be used for predictions in real-time. Chapter 4 offers a methodology for accurate identification of the local permeability of fibrous materials, which is a major property that highly affects the manufacturing process. Finally, chapter 5 offers a summary and a conclusion to the work along with perspectives for future work.

Chapter 1

State of the art

Abstract

The framework of solving forward and inverse problems defined by partial differential equations using Physics-informed neural networks (PINN) is introduced in this chapter. The chapter starts by introducing the basics of feed-forward neural networks in a supervised learning paradigm followed by an introduction to convolutional neural networks, which will be used later in the thesis. The chapter continues by introducing PINN to solve forward and inverse problems defined by a general time-dependent partial differential equation. The final section of the chapter will discuss the recent applications of machine learning in the field of composites manufacturing.

Contents

1.1 Introduction

Over the last few decades, the use of machine learning techniques has significantly increased. Machine learning has proven to be remarkably successful across various domains, including computer vision [1], natural language processing [2], speech recognition [3], medical applications [4], and more. While the fundamental ideas of machine learning were present as early as the mid of the 20^{th} century [5], its widespread success has been realized mainly in recent years, owing to significant advancements in computer architectures that have enabled faster processing speeds. Additionally, the advent of new technologies has made it possible to collect vast amounts of data, which is crucial for constructing highly effective machine learning models.

Despite these achievements, machine learning still faces challenges when it comes to complex physical or engineering applications. One major obstacle is the lack of robustness or convergence guarantee, primarily due to the scarcity of large datasets. Conducting numerous experiments or running extensive physical simulations to gather data for such applications is often impractical or infeasible. However, in engineering and physical systems, valuable knowledge is available in the form of partial differential equations (PDE) that describe the underlying physics, which is not utilized in building machine learning models.

Addressing this disparity between data scarcity and existing physical knowledge, a new concept known as Physics-Informed Neural Networks (PINN) was introduced in 2018 [6]. PINN brings together the available data and the domain-specific knowledge encoded in physics equations to build efficient physics and data-based machine learning models.

In this chapter, the state of the art of PINN will be introduced to solve a forward and an inverse problem governed by a generic PDE. The chapter starts with an introduction to feedforward neural networks within the context of supervised learning. Topics such as parameter initialization, overfitting, automatic differentiation, and optimization algorithms will be thoroughly discussed.

Afterwards, An introduction to convolutional neural networks (CNN), an essential architecture in deep learning, is presented. Subsequently, PINN is introduced as a forward PDE solver of a general time-dependent partial differential equation followed by an example to solve inverse problems using PINN. Topics such as multi-objective optimization and PINN extension to solve coupled problems will be discussed.

The final part of this chapter will focus on the practical applications of machine learning in the field of composites manufacturing. A focus will be directed to the use of machine learning in the context of liquid composite molding (LCM), a family of composite manufacturing processes, and the potential defects associated with it. The attempts made in various studies to leverage machine learning for predicting and preventing these defects in the LCM process will be discussed in brief. Ultimately, the chapter will conclude with a concise summary.

1.2 Supervised learning with multilayer perceptrons

Supervised learning is the most straightforward approach in training machine learning models. It is a type of learning paradigm in which an algorithm learns from a labeled dataset ($\{\mathbf{X}_i, \mathbf{Y}_i\}_{i=1}^{N}$), where N is the number of data points and $(\mathbf{X}_i, \mathbf{Y}_i)$ refers to the i-th data point. It is a labelled dataset since each input data point $\mathbf{X}_i$ is associated with a corresponding output label $\mathbf{Y}_i$. The goal of supervised learning is to build a mapping function $\hat{\mathbf{Y}}$ to make predictions of the output when presented with new, previously unseen data $\mathbf{X}$.

The mapping function $\hat{\mathbf{Y}}$ has parameters (degrees of freedom) that are chosen through minimizing an error function, called the loss function $\mathcal{L}$. This function measures the error between the prediction $\hat{\mathbf{Y}}(\mathbf{X}_i)$ and the labels $\mathbf{Y}_i$ of the existing dataset. A clear choice of the error function can be the mean squared error which is defined as:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\mathbf{Y}}(\mathbf{X_i}) - \mathbf{Y_i}||^2. \tag{1.1}$$

A feed-forward neural network, also referred to as multilayer perceptrons, is a class of functions that are commonly chosen as the mapping function $\hat{\mathbf{Y}}$. The main reason for its wide use is its ability to approximate any function when the proper weights (parameters of the neural network) are used. This conclusion is derived from the universal approximation theorem [7].

A graphical representation of a feed-forward neural network is given in figure 1.1. The structure of such a network is composed of multiple layers; each layer has a certain number of neurons. Each layer is fully connected to the previous and next layer. That is why it is referred to as a fully connected neural network.

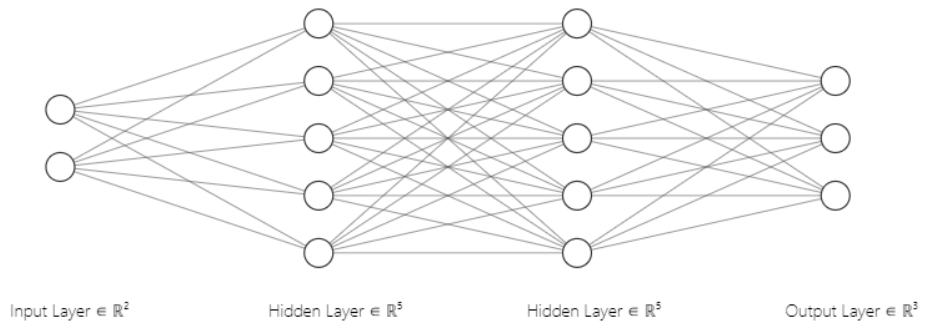

Figure 1.1: A fully connected neural network composed of 2 hidden layers.

The different layers are referred to as $l \in \{0, 1, ..., L\}$. In the deep learning community, the first layer (input layer) is not counted. Therefore, there are 3 layers ($L = 3$) in the network given in figure 1.1. Each layer has a different number of neurons; we refer to the number of neurons in a layer as follows: $\{m_0, m_1, ..., m_L\}$,

for example, m_2 gives the number of neurons in the layer number 2; $m_2 = 5$ in figure 1.1.

The first layer is called the input layer $\mathbf{x}$ that is composed of m_0 neurons, thus $\mathbf{x} \in \mathbb{R}^{m_0}$. The input layer represents the input to the function. The next part of the network is the hidden layers that take the data from the input layer. There can be as many hidden layers and neurons as needed for the learning task; in the example in figure 1.1, the network has 2 hidden layers. Data is moved through the different hidden layers until it is fed to the last layer: the output layer, which is composed of m_L neurons.

Data is transferred from one layer to another in 2 stages. The first stage is a linear transformation. The second one adds non-linearity to the approximation through a non-linear function, the activation function σ. The stages of transformation to move from the input layer to the next hidden layer in our example are given as:

$$\mathbf{Z_1} = \mathbf{W_0}\ \mathbf{X} + \mathbf{b_0} \tag{1.2}$$

$$\mathbf{A_1} = \sigma_1(\mathbf{Z_1}). \tag{1.3}$$

As stated before, the first step is a linear transformation that maps the input $\mathbf{x}$ to the neurons of the next layer $\mathbf{Z_1} \in \mathbb{R}^5$. $\mathbf{W_0} \in \mathbb{R}^{5\times 2}$ is called the weight matrix and $\mathbf{b_0} \in \mathbb{R}^5$ is called the bias vector. The result of this transformation is $\mathbf{Z_1} \in \mathbb{R}^5$. The next step is applying the activation function to add a non-linearity to the transformation resulting in $\mathbf{A_1}$, where σ_1 is the activation function of the first hidden layer. This step is an element-wise application of the non-linear function. The most commonly used activation functions are the relu, hyperbolic tangent, and sigmoid functions. Some activation functions are shown in figure 1.2. It should be noted that different activation functions can be used for different layers.

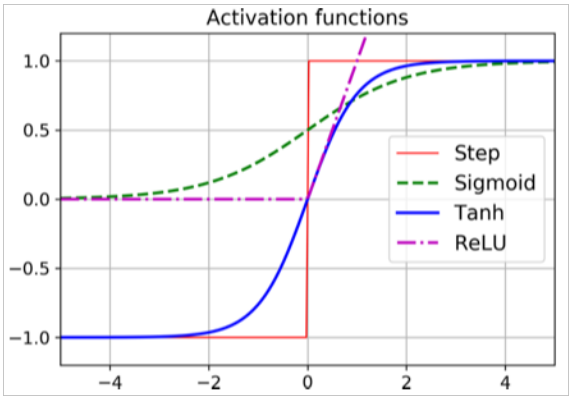

Figure 1.2: Activation functions σ

Similar transformations are made for the other layers until the output $\hat{\mathbf{Y}}$ is reached. The full model or approximation for the network in figure 1.1 can be written as:

$$\hat{\mathbf{Y}} = \sigma_3(\mathbf{W_2}(\sigma_2(\mathbf{W_1}(\sigma_1(\mathbf{W_0}\mathbf{X} + \mathbf{b_0})) + \mathbf{b_1})) + \mathbf{b_2}). \tag{1.4}$$

For simpler representation, all the model degrees of freedom (weights and biases matrices) are stacked in one vector γ. To find the best model parameters that fit the given data, the loss function defined in equation 1.1 is minimized with respect to γ. The minimization problem is defined as:

$$\min_{\gamma} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\mathbf{Y}}(\mathbf{X}_i; \gamma) - \mathbf{Y}_i||^2. \tag{1.5}$$

Several algorithms are used to solve this minimization problem; some of the most used techniques are discussed in the next section.

1.2.1 Optimization Algorithms

Many existing minimization algorithms for optimizing machine learning models are based on gradient descent [8]. Gradient descent is a first-order iterative optimization method used to minimize differentiable functions. The basic idea is to iteratively update the model parameters in the opposite direction of the gradient of the loss function at the current point, as it indicates the steepest descent. The gradient descent step is defined as follows:

$$\gamma_{n+1} = \gamma_n - \lambda \nabla \mathcal{L}(\gamma_n), \tag{1.6}$$

where γ_n is the current point, γ_{n+1} the new point, λ is the learning rate that defines the size of the step taken in the gradient direction, and $\nabla \mathcal{L}(\gamma_n)$ is the gradient of the loss function with respect to γ defined at γ_n.

If the whole dataset is used at once to get the gradient of the loss function and update the model parameters, this is called Batch minimization or training, or Batch gradient descent. While if only one data point is used to get the gradient and update the parameters, the algorithm becomes stochastic gradient descent [9].

Batch gradient descent usually has stable convergence as compared to stochastic gradient descent which is characterized by noisy convergence and in some cases might not converge to the global minimum. However, Batch gradient descent suffers from being computationally expensive and requires high memory storage. To get the best of the two, mini-batch gradient descent is used [9]. It is based on dividing the dataset into small batches called mini-batches, and then each batch is used to get the loss gradient and update the model parameters. This way, the size of the mini-batches is tuned according to the memory requirement and to make the convergence stable.

The idea of mini-batches gives rise to the term epoch. An epoch refers to the pass of the minimization algorithm over the whole dataset. It should not be confused with iterations of the minimization algorithm which refers to updates of the network parameters. For example, if the training data is divided into 10 mini-batches; one epoch will be equal to 10 iterations (gradient descent steps).

Mini-batches can also be applied to other minimization algorithms other than gradient descent. Other more used first-order algorithms which are variants of gradient descent include Momentum [10], RMS prop [11] or Adam [12] algorithms. These

algorithms are frequently used and are available in most of the machine learning libraries.

Second-order minimization algorithms can be a good alternative to gradient-based algorithms. These algorithms take into account not only the first-order derivative (gradient) of the loss function with respect to the model parameters but also the second-order derivatives (Hessian matrix or approximations) to make more informed updates to the parameters. The Hessian matrix gives information about the shape of the loss function making the parameters update more informed. This can lead to faster convergence and improved optimization. Most of the second order algorithms are based on Newton's method which is a classic second-order optimization algorithm that updates the parameters using the inverse of the Hessian matrix. The update rule can be expressed as follows:

$$\gamma_{n+1} = \gamma_n - \lambda H^{-1} \cdot \nabla \mathcal{L}(\gamma_n), \tag{1.7}$$

where H^{-1} is the inverse of the Hessian matrix.

Obtaining the inverse of the Hessian matrix is a complex process and is not practical to obtain exactly for average size or big models. The used algorithms in practice are based on approximating the inverse of the Hessian matrix such as BFGS algorithm [13]. Another issue occurs because of the high storage needs of second order algorithms that is why other methods are based on getting a sparse approximation to the inverse Hessian matrix or saving several vectors that are used for the inverse Hessian matrix approximation. An example to such algorithms are the limited memory version l-BFGS [14].

Despite there better converge compared to first-order optimizes, second order algorithms are not commonly used due to the higher storage requirements and computational cost which is not suitable for large scale machine learning tasks.

1.2.2 Automatic differentiation

To perform the training using any optimization algorithm, the derivative of the loss function with respect to the model parameters is needed in the process. For complex models, the analytical derivative is complicated to obtain, thus, other ways are used to get the derivative.

One way of getting the derivative is numerical differentiation. It works by calculating the loss function at the current point γ_0 and at $\gamma_0 + \Delta\gamma$, where $\Delta\gamma$ is a small value. Then, the derivative is approximated using finite differences as:

$$\frac{\partial \mathcal{L}}{\partial \gamma} \approx \frac{\mathcal{L}(\gamma_0 + \Delta\gamma) - \mathcal{L}(\gamma_0)}{\Delta\gamma} \tag{1.8}$$

There are two main drawbacks to using numerical differentiation. The first is the possible small round-off errors in the calculation of the gradient which can hinder the optimization process. The second and main drawback is the slow computation of the gradient as the number of basic mathematical operations is big, which is the

case in neural networks. That is why numerical differentiation is not commonly used in deep learning.

Automatic differentiation (AD) offers a solution to the previously mentioned drawbacks [15]. It is based on the idea that any complex function can be represented as a chain of basic arithmetic, trigonometric, or logarithmic operations. The derivative can be constructed as a computational graph and the chain rule can be used to accurately and efficiently calculate the derivative.

An example of the computation graph of a simple function is given in figure 1.3. As we can see, the function is broken down into simpler basic operations in which the derivative of each operation is stored in the forward step. Then, the derivative can be calculated with the chain rule using the already saved values.

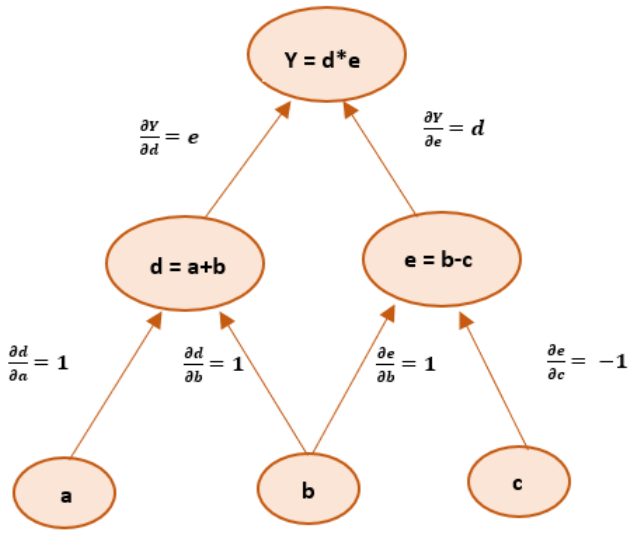

Figure 1.3: An example of a simple computation graph used in the automatic differentiation process.

1.2.3 Parameters initialization

Initializing the neural network parameters plays a crucial role in the training and performance of neural networks. The initial values assigned to these parameters determine the starting point for the optimization process. If the parameters are poorly initialized, it can lead to the vanishing or exploding gradients problem, which occurs when the gradients used to update the network are extremely small or large. The goal of parameter initialization is to provide a sensible starting point that allows the network to learn effectively.

One of the simplest initialization methods is random initialization, where the parameters are initialized with random values drawn from a distribution. This approach provides randomness and helps break the symmetry among neurons in the network.

Another method is called Xavier initialization, also known as Glorot initialization [16]. This is a widely used technique that aims to address the vanishing and exploding gradient problems. It initializes the parameters by drawing values from a distribution with zero mean and variance that depends on the number of incoming and outgoing connections to a neuron. The distribution is chosen to ensure that the variance of the activations remains constant across layers.

Another technique is transfer learning [17]. In this approach, parameters are initialized with values obtained from a previously trained model on a related task or dataset. This initialization leverages the knowledge gained from the pre-trained model and can significantly speed up convergence and improve performance, especially when training data is limited.

1.2.4 Overfitting and Underfitting

The aim of supervised learning is not just to fit the data used for the training (optimization process) but also to be able to make good predictions when given unseen data $\mathbf{X}$. Minimizing the loss function on the training set does not usually ensure producing a good model. That is why the given dataset is usually split into 3 different sets: a training set used for the optimization, a validation set used for tuning the model to make sure it can generalize to data that was not directly used to train the model, and a testing set used to make a final test to the model on unseen data.

If the model is performing extremely well on the training set but bad on the validation set, it can be concluded that the model will not perform well on unseen data and it is said that the model is overfitting. If the model neither performs well on the training or validation sets, the model is said to be underfitting. If the model performs well on both the training and validation sets, it is neither underfitting or overfitting and it can be assessed on the testing set before being deployed in a real application. Figure 1.4 is an example of overfitting and underfitting on classification tasks.

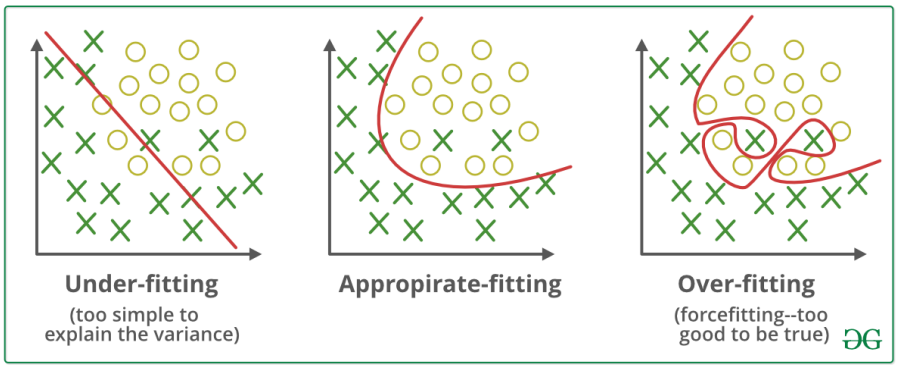

Figure 1.4: Example of overfitting and underfitting on a simple classification task.

The underfitting problem can be solved, most of the times, by increasing the complexity of the used neural network (more hidden layers and neurons). However,

more complex techniques are used to solve the overfitting issues. The most commonly used techniques are:

1. More data:

   The first way to avoid overfitting is to collect more data which helps make the model able to generalize and less affected by outliers. This can be performed by making more experiments or simulations in physical-related processes. Data augmentation can also be used to increase the dataset [18]. It is achieved by modifying the already existing dataset. If the data used are images, this can be achieved by applying random rotations and cropping to the used images.

2. Regularization:

   Regularization works by adding a penalty term to the loss function during training, which discourages the model from becoming too complex or relying too much on specific features in the training data. This penalty encourages the model to generalize better to unseen data, improving its ability to make accurate predictions on new examples. The added term to the loss function makes sure that some of the model parameters are close to zero or zero making the model spare, thus, less complex.

   One of the most commonly used regularization techniques is the L_1 regularization [19], where the modified loss function can be written as:

   $$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} ||\hat{\mathbf{Y}}(\mathbf{X}_i;\gamma) - \mathbf{Y}_i||^2 + \epsilon \sum_{j=1}^{p} |\gamma_j| \tag{1.9}$$

   where ϵ is a regularization hyperparameter to be tuned, γ_j the components of γ, and p the length of γ. Another technique is L_2 regularization [19] where the modified loss function can be written as:

   $$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} ||\hat{\mathbf{Y}}(\mathbf{X}_i;\gamma) - \mathbf{Y}_i||^2 + \epsilon \sum_{j=1}^{p} \gamma_j^2 \tag{1.10}$$

3. Early stopping:

   Early stopping is another technique to avoid overfitting which is based on stopping the training process once the model starts to overfit [20]. This can be done by monitoring the loss function on both the training and validation datasets; once the loss on the validation set starts to deviate and increase while the training loss keeps decreasing, it means that the model starts to overfit and it is a good point to stop. Figure 1.5 explains the idea of early stopping. It shows an example of the loss function evolution with the number of iterations over the training and validation datasets for an overfitting case and the early stopping point.

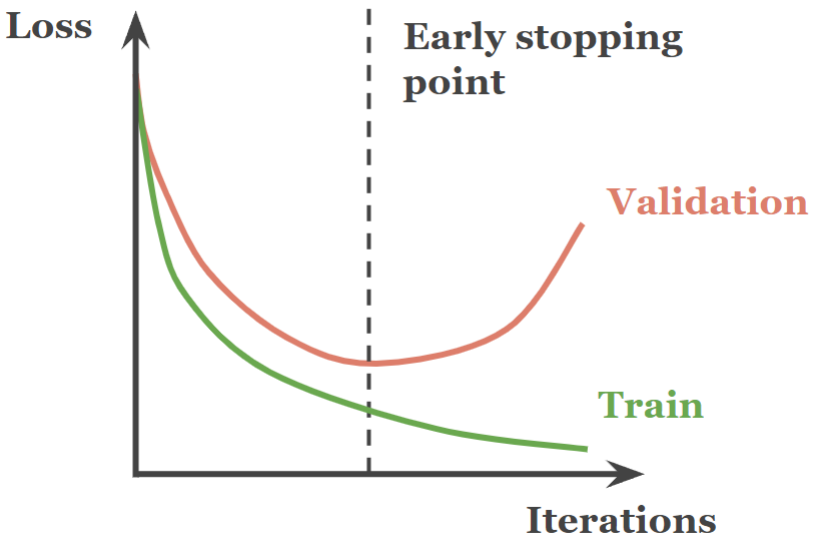

Figure 1.5: An example of the early stopping technique shows the evolution of the loss function over the training and validation datasets and the early stopping point.

1.3 Convolutional neural networks

When the input $\mathbf{X}$ to the fully connected network is an image (matrix form), one has to reshape the image into a 1D vector to be able to feed it into the network. For example, a color image of size 32×32 with three channels (RGB) would be reshaped into a vector of size 1×3072 ($32 \times 32 \times 3$). The size of the input layer would be huge leading to a big weight matrix since every neuron connects to every neuron in the preceding and succeeding layers. Therefore, using fully connected layers for images would require an immense number of parameters, leading to high computational complexity, memory requirements, and a heightened risk of overfitting.

CNNs have become popular and commonly used for processing images. Their key advantage over fully connected neural networks is their ability to exploit local connectivity and weight sharing. CNNs excel at capturing spatial hierarchies, extracting meaningful features, and reducing the parameter count compared to fully connected networks. This makes CNNs ideal for tasks like image recognition, object detection, and image segmentation.

CNNs are specifically designed to process data with a grid-like structure, such as images, by leveraging the concept of convolution. At the core of CNNs are convolutional layers, which perform the convolution operation on the input data. Convolution is a mathematical operation that combines the input with a set of learnable filters or kernels, enabling the network to extract local patterns and spatial relationships from the data.

The operation involves sliding the filter across the input, computing the dot product between the filter and the overlapped region of the input at each location.

The resulting values form the feature map after which a nonlinear activation function is usually applied. The convolution operation is illustrated in figure 1.6.

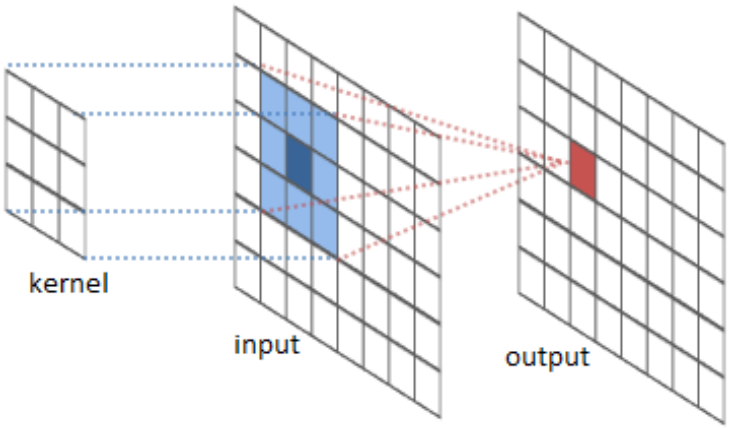

Figure 1.6: Basic convolution operation [21].

The pooling layer is another important layer that is used to reduce the spatial dimensions of the feature maps while retaining important information. They achieve this by downsampling the feature maps through operations like max pooling or average pooling, which extract the most dominant features within local regions. It should be noted that these layers have no learnable parameters. The max and average pooling operations are explained in figure 1.7.

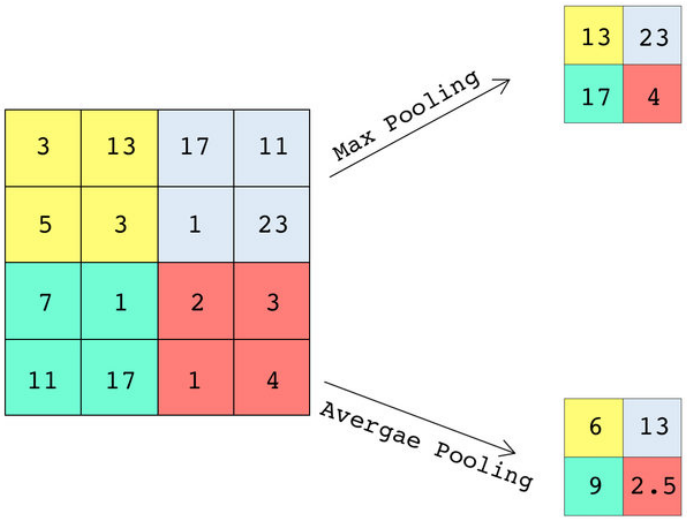

Figure 1.7: Max and average pooling operations [22].

After several convolutional and pooling layers, the resulting features are often flattened and passed through fully connected layers. These layers resemble traditional neural networks, where each neuron is connected to every neuron in the previous layer. The fully connected layers integrate the spatial information captured by earlier layers and map it to the desired output, such as specific class labels or regression

values.

Training a CNN involves optimizing its parameters, which include the weights of the filters and the parameters of the fully connected layers. This optimization is achieved using backpropagation, where the gradient of a suitable loss function with respect to the network's parameters is computed. One of the simplest and most famous CNN networks is the LeNet-5 architecture introduced in 1998 [23]. The network is introduced to identify numbers from 0 to 9 from handwritten pictures. The architecture is shown in figure 1.8.

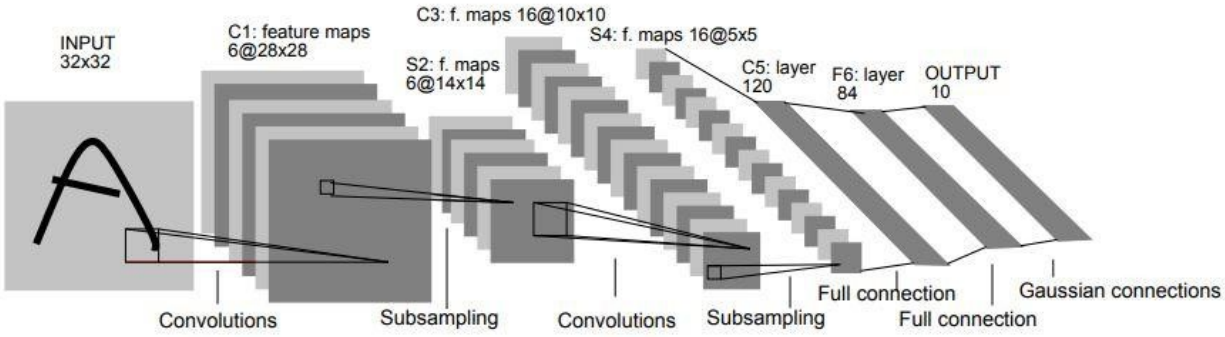

Figure 1.8: LeNet-5 architecture [23].

Form figure 1.8, we can see that the input image has a shape of (32×32). The first layer is a convolutional layer having 6 filters of size (5×5). Average pooling is, then, applied to the output. Another convolutional layer having 16 filters of size 5×5 is applied, followed by another average pooling layer. Finally, the output is flattened and fully connected layers are applied ending with 10 neurons for the output corresponding to the 10 digits from 0 to 9.

1.4 Physics-informed neural networks

To obtain a good model, neural networks are usually supplied with big amounts of data. In engineering and science applications, such amounts are normally hard to obtain whether because experiments are hard and time-consuming or computer simulations are costly and also time-consuming. This is the reason explaining that the application of neural networks in scientific applications is limited. Physics-Informed Neural Networks (PINN) offer a way to deal with this low data regime.

PINN is a class of machine learning models that combine the strengths of traditional neural networks and the governing laws of physics. They leverage the expressive power of neural networks to represent complex functions while ensuring that the solutions generated are consistent with the underlying physical principles. The method was introduced by Raissi et. al [6] to solve forward and inverse problems represented by partial differential equations.

1.4.1 Solving Forward problems

PINN is introduced in this section to solve a general nonlinear partial differential equation that has the form of

$$\begin{aligned} u_t + \mathcal{N}(u) = 0, \quad \mathbf{x} \in \Omega, \quad t \in [0,T], \\ u(0,\mathbf{x}) = u_i, \quad \mathbf{x} \in \Omega, \\ u(t,\mathbf{x}) = u_D, \quad \mathbf{x} \in \Gamma_D, \quad t \in [0,T], \\ B(u(t,\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma_N, \quad t \in [0,T], \end{aligned} \tag{1.11}$$

where u_t is the time derivative and $\mathcal{N}$ is a nonlinear differential operator applied to the solution $u(t,\mathbf{x})$, u_i is the initial condition and u_D is a Dirichlet boundary condition defined over Γ_D. B is an operator defining the Neumann boundary condition defined over Γ_N

The first step in the PINN solution is to choose the approximation space for u as a feed forward neural network denoted by $\hat{u}$.

$$u \approx \hat{u}(t,\mathbf{x};\gamma) \tag{1.12}$$

where γ denotes the unknown parameters (weights and biases) of the neural network.

Afterwards, we define f through automatic differentiation as follows:

$$f := \hat{u}_t + \mathcal{N}(\hat{u}) \tag{1.13}$$

The loss function is then defined as:

$$\mathcal{L} = \lambda_0 \, \mathcal{L}_0 + \lambda_D \, \mathcal{L}_D + \lambda_N \, \mathcal{L}_N + \lambda_r \, \mathcal{L}_r \tag{1.14}$$

where λ_i are the weights for each loss term which play an important role in the optimization process and:

$$\mathcal{L}_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} r_0^2(t_0^i, \mathbf{x}_0^i) = \frac{1}{N_0} \sum_{i=1}^{N_0} ||\hat{u}(t_0^i, \mathbf{x}_0^i) - u_0^i||^2 \tag{1.15}$$

$$\mathcal{L}_D = \frac{1}{N_D} \sum_{i=1}^{N_D} r_D^2(t_D^i, \mathbf{x}_D^i) = \frac{1}{N_D} \sum_{i=1}^{N_D} ||\hat{u}(t_D^i, \mathbf{x}_D^i) - u_D^i||^2 \tag{1.16}$$

$$\mathcal{L}_N = \frac{1}{N_N} \sum_{i=1}^{N_N} r_N^2(t_N^i, \mathbf{x}_N^i) = \frac{1}{N_N} \sum_{i=1}^{N_N} ||B(\hat{u}(t_N^i, \mathbf{x}_N^i)) - f_N^i||^2 \tag{1.17}$$

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} ||r(t_r^i, \mathbf{x}_r^i)||^2 = \frac{1}{N_r} \sum_{i=1}^{N_r} ||\hat{u}_t + \mu L(\hat{u})||^2_{(t_r^i, \mathbf{x}_r^i)} \tag{1.18}$$

$$\tag{1.19}$$

$\mathcal{L}_i$ respectively are the losses representing the initial condition, Dirichlet, Neumann boundary conditions, and the PDE residual.

where $\{x_i^j, t_i^j, u_i^j\}_{j=1}^{N_i}$ denotes points defining the initial condition, $\{x_D^j, t_D^j, u_D^j\}_{j=1}^{N_i}$ the points defining the Dirichlet boundary condition, and $\{x_r^i, t_r^i\}_{i=1}^{N_r}$ the collocation (residual) points where the differential equation is to be enforced. That makes the first term of Eq. 1.14 correspond to the initial condition, the second term refers to the Dirichlet boundary condition, while, the last term is related to imposing the differential equation in the domain.

The different terms of the loss function are weighted differently through λ_i, λ_D, and λ_r. This is done since the different terms can have different scales and weighting them is important for the minimization process. Finding the appropriate weights is not an easy task in the process. Some techniques recently introduced in the literature are discussed in a future section 1.4.4.

A solution to the problem can then be obtained by solving the following minimization problem:

$$\hat{\gamma} = \min_{\gamma} \mathcal{L} \tag{1.20}$$

1.4.2 Solving Inverse problems

The same problem defined by equations 1.11 is used to define an inverse problem. In this case, the nonlinear differential operator $\mathcal{N}$ depends on a parameter μ whose value is unknown and homogeneous in space and time, meaning that the unknown is just one value. The PDE defining the problem can be rewritten as:

$$u_t + \mu\mathcal{N}(u) = 0. \tag{1.21}$$

In addition to the known initial and boundary conditions, the solution is known at a few scattered points in space and time, which can represent sensor data in a real physical process. The aim is to get a solution to the PDE and an estimation of the unknown μ using the known physics and scattered data.

The strategy of PINN is to add an extra term to the loss function equation 1.14 that represents the mismatch in the scattered data. The term can be written as:

$$\mathcal{L}_{data} = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} ||\hat{u}(t_{data}^i, \mathbf{x}_{data}^i) - u_{data}^i||^2 \tag{1.22}$$

where $\{x_{data}^j, t_{data}^j, u_{data}^j\}_{j=1}^{N_{data}}$ denotes points defining the scattered data.

A solution to the PDE u and an estimation of μ can then be obtained by minimizing the modified loss function after adding the extra term with respect to the neural network parameters and μ.

$$\hat{\gamma}, \hat{\mu} = \min_{\gamma,\mu} \mathcal{L} \tag{1.23}$$

1.4.3 Extension to coupled partial differential equations

Using PINN, the extension to solving several coupled partial differential equations is natural. If several fields need to be approximated, a network with several outputs can be used, or several separate networks. AD is used to calculate the residuals for the different equations. Finally, the loss function will be composed of more terms related to the different residual fields along with the initial and boundary conditions. This extension will be clear in the next chapter.

1.4.4 Multi-objective optimization

The loss function in PINN is usually composed of several terms representing the residuals of the differential equations to be solved, the initial and boundary conditions, or terms representing known data. The terms usually have different magnitudes which creates a difficulty in the minimization process. Minimizing such composite loss function is known as multi-objective optimization since the minimization process has to do several objectives simultaneously while the different terms will be competing.

A naive way to perform such a task is to weigh each term differently so that all terms have similar magnitude, thus, will be minimized in the same manner. Choosing the weights can be done using a trial and error technique. One can start with equal weights and gradually modify them based on the results obtained. Iteratively refining the weights based on trial and error can help find the desired balance and achieve the objectives effectively.

Non-dimensionalization of the full system of equations under study can help fix the issue. It can aid create different terms having to start the minimization with comparable magnitudes; thus making the training easier. Non-dimensionalization has shown to be efficient in many studies [24; 25], however, in some cases, non-dimensionalization may not be enough, and other methods need to be used.

A way to reduce the number of terms in the loss function, thus simplifying the optimization task, is to use hard enforcement of initial and boundary conditions. The way PINN is satisfying the initial and boundary condition is called a "soft" way of enforcing the conditions. This soft enforcement of initial and boundary conditions has several drawbacks. First, there is no guarantee of the accuracy of the enforcement of the conditions. Second, the optimization performance is related to the relative importance of each term which is dependent on the weighting coefficients which is difficult to choose. Alternatively, the initial and boundary conditions can be satisfied in a "hard" way by having a particular solution satisfying the conditions exactly at the boundaries. This method makes sure that the conditions are automatically satisfied and the optimization performance is enhanced since there is no more an issue of relative importance between terms of the loss. Several research were conducted making use of the "hard" way of enforcing the initial and boundary conditions [26–28]. The solution is formulated as follows:

$$\hat{u}(x,t) = u^p(x,t) + D(x,t)\ u^{NN}(x,t) \tag{1.24}$$

where u^p is a particular solution that satisfies the initial and boundary conditions. D is a globally defined smooth function that takes a value of zero on the spatio-temporal boundary where conditions are defined and increase away from the boundary. u^{NN} is the general solution in the domain approximated with a neural network. For simple geometries, u^p and D can have analytical forms, however, for more complicated problems, u^p and D can be approximated with neural networks that are pre-trained before solving the problem. This way was used by [26; 27].

Another similar way of using the "hard" enforcement, introduced by [28], is to use a function H that takes the value 0 at the boundaries and 1 inside the domain, instead of using the D function. This function is defined as follows:

$$H(x,t) = \begin{cases} 1 - \cos\left(d(x,t)\pi/\epsilon\right) & d < \epsilon \\ 1 & d \geq \epsilon \end{cases} \tag{1.25}$$

where d is the distance to the boundary and ϵ is an artificial thickness of the boundary. Through this definition, the solution will take the form

$$\hat{u}(x,t) = u^p(x,t)(1 - H(x,t)) + H(x,t)\ u^{NN}(x,t) \tag{1.26}$$

This means that at the boundaries where H is 0, the solution will only be u^p which satisfies the initial and boundary conditions exactly. However, inside the domain where H is 1, the solution will be only u^{NN}.

The modified loss function will then have only one term related to satisfying the differential equation in the domain, while the initial/ boundary conditions are implicitly satisfied because of the previous imposition in the particular solution.

1.5 Applications of Machine Learning on Composites Manufacturing

1.5.1 Resin Transfer Molding

In this subsection, an introduction to Resin Transfer Molding (RTM), a manufacturing process that falls under the category of LCM processes, is introduced. LCM processes have a common feature of filling a cavity with dry reinforcement then the liquid resin is injected into the cavity. In this section, the RTM manufacturing process will be discussed [29].

In general, RTM process can be subdivided into 3 main steps: preforming, resin injection, and in-mold cure/polymerization. After these steps are done, machining and finishing can take place for the part. A summary of the RTM process is shown in figure 1.9.

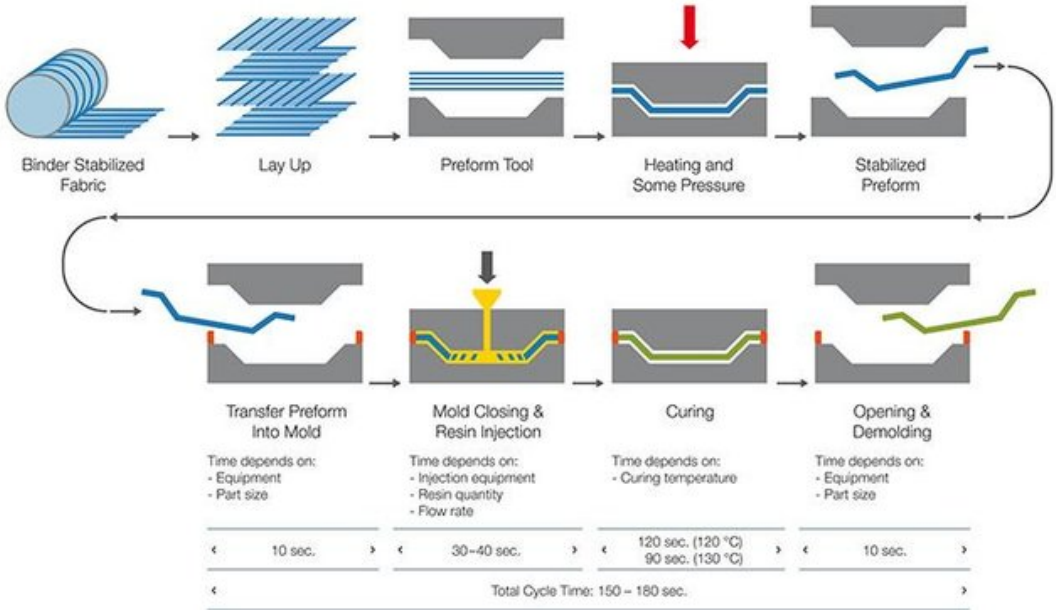

Figure 1.9: Resin transfer molding process steps [30].

In the preforming step, the dry reinforcement is tailored to have the shape of the mold. A preform binder is usually used to make the preform stiff enough to be stacked and handled before injection and to force the fibers to stay in their position and orientation during injection. Once this process is finished, the preform is placed into the mold cavity which is then closed. The preform is usually slightly compressed by the mold. This step is done to speed up the process, improve the quality and reduce part-to-part variations. Several methods can be used depending on the mold complexity and the required mechanical performance. The used preforming methods are: cut and paste, spray-up or thermoforming, for instance.

The next step is the resin injection. The resin is injected in the mold under constant pressure or constant flow rate until the preform is fully impregnated. During this step, resin pushes the air outside and an interface between the resin and air exists referred to as the flow front. There are three main strategies for mold filling: point injection, edge injection, and peripheral injection. In point injection, the resin is introduced in the center of the part through a port. The resin will flow radially venting air at the edge of the part. For edge injection, the resin is injected through a lineal inlet from one edge of the part. The flow will be almost unidirectional venting the air to the other edge. Finally, the peripheral injection introduces the resin in a distribution channel around the periphery of the part. The flow will go inward venting the air at the center of the part. Injection parameters such as: injection pressure, viscosity, or temperature for example need to be carefully chosen to avoid problems such as: dry spots or fiber washing.

The last step in the process is the in-mold polymerization or crystallization for thermosets. The polymerization involves chemical transformation of the resin liquid into a gel state (therefore an increase of the viscosity) and then to become a hard solid. Polymerization usually takes place during the injection process and after it,

however, the polymerization (increasing viscosity) should be slow during injection to make sure the injection step is done correctly and then, the reaction can become faster to reach a complete polymerization in a short time.

1.5.2 Issues during LCM processes

During LCM processes, several issues can arise if the process is not well-designed or lacks proper control. Understanding the reasons behind these issues and their consequences is crucial for preventing them and determining acceptable tolerances. Some common defects that can occur during an LCM process are early gelation of the resin, the formation of dry spots, fiber-wash out, and fiber misalignment.

For instance, one of the issues is early gelation of the resin [31]. Gelation is the first solidification stage in which the viscosity of the resin highly increases stopping it from filling the mold. This happens if the mold is heated more than needed. As a consequence, the polymerization starts earlier than planned and then the injection process is incomplete because the resin has solidified quicker. Figure 1.10 shows the viscosity vs. temperature for a thermosetting epoxy resin.

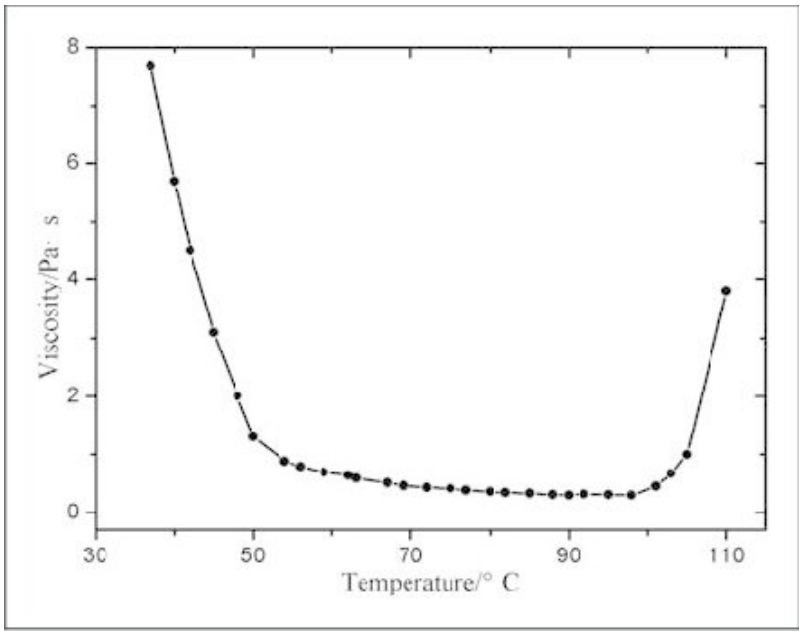

Figure 1.10: Viscosity vs. temperature of a thermosetting epxoy resin [32].

From figure 1.10, one can see that the viscosity decreases as the temperature increases which allows for a faster flow in the mold, however, at some point, the viscosity sharply increases (gelation point) due to a chemical reaction starts to taking place.

Another common issue is the formation of dry spots during the injection. Dry spots are air pockets surrounded by resin that happens during injection processes. This is a critical problem since mechanical properties such as: shear and tensile strength are degraded with increasing the void content of the part [33; 34]. For this reason, depending on the application, products with more than 1 to 5% of void

content can be rejected. This issue could occur when the geometry of the part is complicated so that the flow pattern is not regular, and thus dry spots are common in this case [35]. This issue is also common for multiple gates injection strategies. If the gates are not well-designed in location and to open at the correct times, dry spots will be created [36]. Several dry sports are shown in figure 1.11

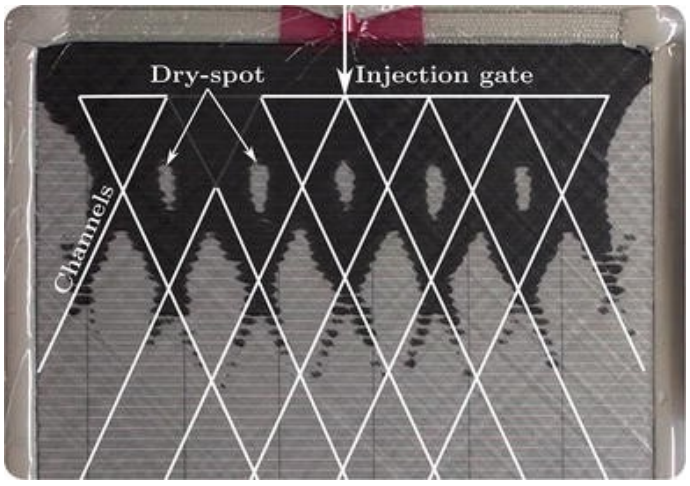

Figure 1.11: Several dry spots appearing in an LCM process [37].

Another main defect that is common in LCM is race tracking. Race tracking occurs when there is a gap between the preform and the mold wall. This misplacement leads to faster resin flow in this gap leading to an unexpected filling pattern. As a consequence, the resin might reach the outlet before completely saturating the preform, leading to the creation of dry spots. The local permeability in the race tracking region depends on the size of the gap. Figure 1.12 shows an example of race tracking.

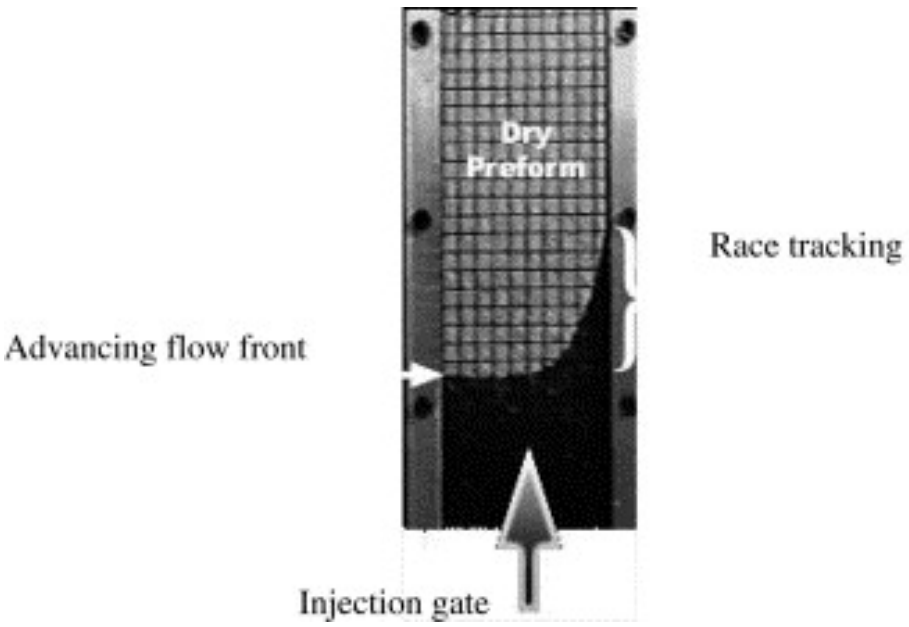

Figure 1.12: An example of the race tracking defect in resin transfer molding [38]

Another problem that might take place is fiber-wash out. Fiber-wash out is the displacement of fibers from their original position during the injection. This issue is common for high-pressure injection processes; the inlet pressure is so high that the fibers near the inlet are usually displaced [39]. This leads to localized weak points in the part where there are almost no fibers and it is usually near the inlet. Figure 1.13 shows deformed tows near the inlet due to high pressure.

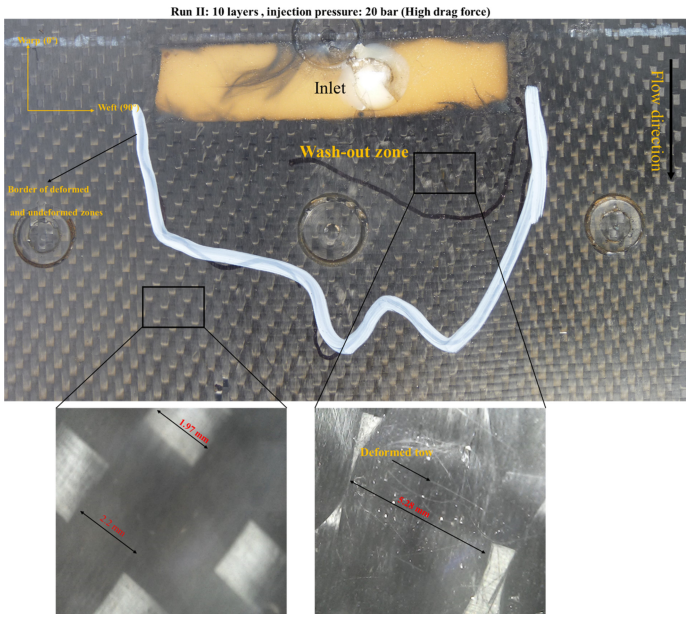

Figure 1.13: Fibre-wash out example due to high pressure [39].

Fiber misalignment can also be seen as a type of defect [39]. It happens when the fibers should be aligned in a specific direction and small angular misalignment takes place during the performing or placement of the fibers in the mold. This misalignment can lead to high deterioration of the desired properties in the direction of interest.

1.5.3 Machine learning applications

Machine learning recently gained considerable attention in the field of composite manufacturing. Its application in the field can vary from process optimization to detection of defects in real-time allowing manufacturers to intervene and take corrective actions to material characterization. In this section, we offer a literature review with the most promising research performed in this area related to resin transfer molding manufacturing process.

In [40], the authors used feed-forward neural networks to make predictions of the location of the flow front. The input to the network is the current flow front location and the flow rate, while the output is the new location of the flow front after 1 second. The data used for the training is collected from numerical models. The trained network is used in a controller regime to change the flow rate to avoid the formation of dry spots. The framework is built for a two-dimensional unidirectional

study case and is shown in figure 1.14.

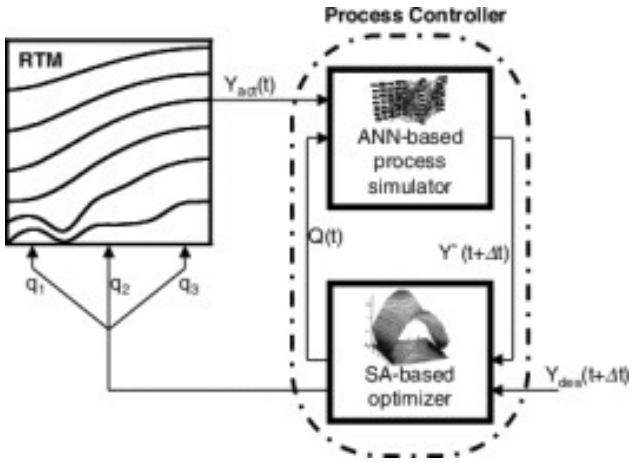

Figure 1.14: The framework introduced by [40] to avoid the formation of dry spots. The ANN-based simulator is a trained neural network taking the current flow front and the flow rate as inputs and outputs the expected flow front location after 1 second. The predicted output along with the desired flow front location is fed to an optimizer that changes the flow rate according to the desired front location.

Results from [40] of the predicted flow front location using the neural networks are shown in figure 1.15. The results show agreement when compared with a numerical solver.

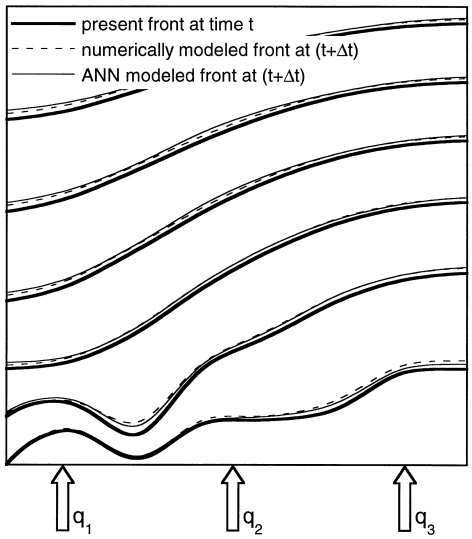

Figure 1.15: The flow front location predictions of the framework introduced by [40]. The predictions of the neural network are compared to predictions by a numerical method.

In [41], a framework is developed to detect the location, size, and permeability of a defect from pressure sensors. Data in the form of 9 pressure sensors are collected from 3k numerical simulations where the location, size, and permeability of defects

are varied. The pressure sensor data is stored as an image footprint of the pressure. A CNN is built to perform the task where the input is the image representing the pressure data, and the output is the location, size, and defect permeability. A visual representation of the used CNN is shown in figure 1.16

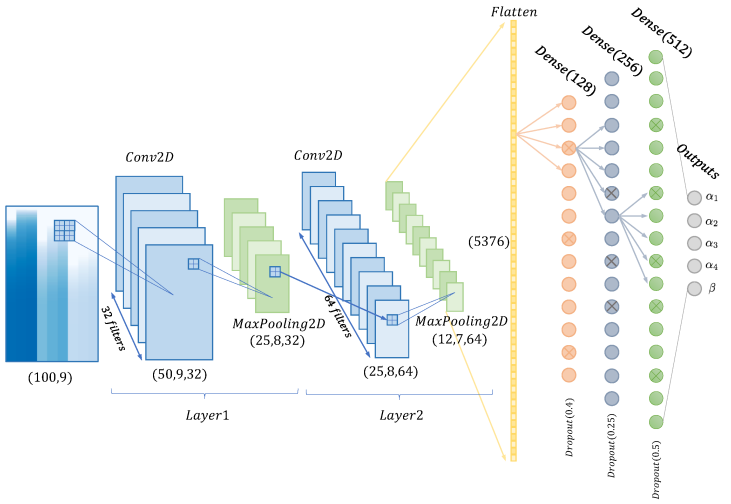

Figure 1.16: The architecture of the convolutional neural network used by [41] to predict the location and size of defects using pressure sensor data. The input to the network is an image representing the pressure sensor data and the output is the location and size of the defect.

The predictions of the location and size of the defect in [41] show close agreement with the ground truth when 3×3 pressure sensor grid was used as shown in figure 1.17.

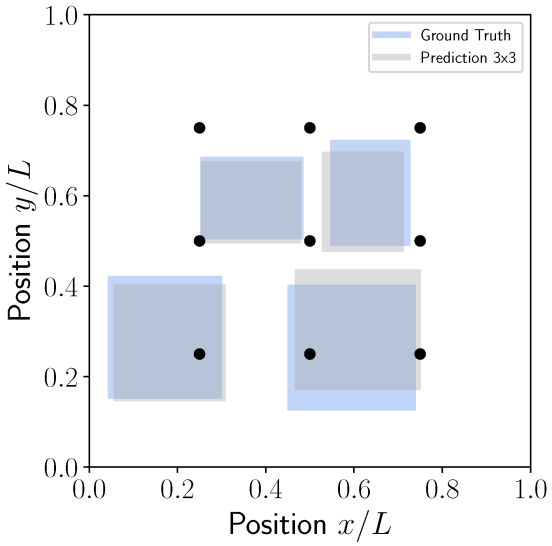

Figure 1.17: The prediction of the location and size of the defect done by a convolutional neural network introduced in [41] as compared to the ground truth. A 3×3 pressure sensor grid was used as an input to the network.

The authors of [42] developed a framework based on convolutional neural networks to predict the existence of dry spots from pressure sensors. The developed network

is split into two parts: a deconvolutional/convolutional part to generate flow front images from pressure sensors and a classical convolutional network to predict whether a dry spot exists or not from the flow front image. The data used for training is produced using 36k numerical simulations of a two-dimensional central injection problem with varying local permeabilities representing defects. The used neural network is shown in figure 1.18.

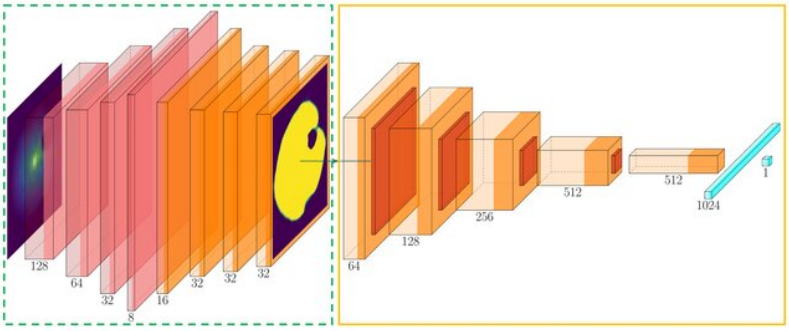

Figure 1.18: The neural network architecture used by [42] to predict the existence of dry spots. The first part is a deconvolutional/convolutional network, which takes an image representing pressure sensor data as an input and outputs a flow front image. That generated image is fed into a convolutional network for binary classification to predict whether a dry spot exists or not.

An example of the generated flow front image from the deconvolutional/convolutional part is shown in figure 1.19.

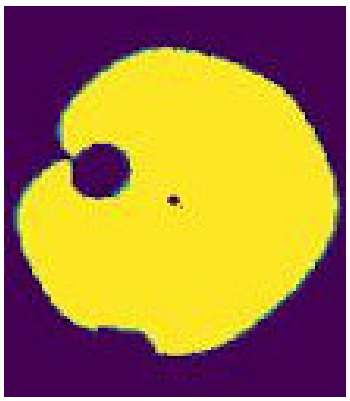

Figure 1.19: An example of the generated flow front image from the deconvolutional/convolutional part of the network architecture introduced by [42]. The input to the network is an image representing pressure sensor data.

The authors of [42] obtained 91.7% accuracy using 1140 pressure sensors, 83.7% using 80 sensors, and 75.22% using 20 sensors. Even though the framework to generate flow front images from sensors is unique and promising; it is not practical to have such a high number of sensors in a real process.

1.6 Summary

In this chapter, we have seen:

- a detailed description of fully connected neural networks in a supervised learning paradigm.
- important topics in machine learning including optimization algorithms, automatic differentiation, parameters' initialization, and methods to avoid overfitting.
- basics of convolutional neural networks.
- an introduction and description of PINN to solve forward and inverse problems.
- several applications of machine learning in composites manufacturing.

Transition

We have seen in this chapter, the potential of PINN to merge Physics and data knowledge in order to solve problems. In the next chapter, we will see the application of PINN to liquid transfer molding. The injection stage of the process will be in consideration; the process involves a two-phase flow in fibrous porous media.

1.7 Bibliography

[1] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[2] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[3] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[4] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[5] A.G. Ivakhnenko and V.G. Lapa. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

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

[7] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[8] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254):10, 2012.

[9] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[10] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[11] G Hinton. Neural networks for machine learning online course.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[14] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[15] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[17] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[18] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *2016 international conference on digital image computing: techniques and applications (DICTA)*, pages 1–6. IEEE, 2016.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Regularization for deep learning. *Deep learning*, pages 216–261, 2016.

[20] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[21] Jaswanth Sreeram, Stephan Herhut, and Lindsey Kuper. Bringing parallelism to the web with river trail.

[22] Nura Aljaafari. *Ichthyoplankton classification tool using Generative Adversarial Networks and transfer learning*. PhD thesis, 2018.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

[25] Chen Xu, Ba Trung Cao, Yong Yuan, and Günther Meschke. Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Computer Methods in Applied Mechanics and Engineering*, 405:115852, 2023.

[26] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[27] Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

[28] Qiming Zhu, Zeliang Liu, and Jinhui Yan. Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks. *Computational Mechanics*, 67(2):619–635, 2021.

[29] Raju S Dave and Alfred C Loos. *Processing of composites*. Hanser Publishers Munich, 2000.

[30] Arzu Ahmadova. Numerical modelling of porosity generation, movement, and compaction during the rtm process. 2018.

[31] H Farrahinia, A Shojaei, and MR Pishvaie. Numerical simulation and optimization of resin transfer molding cycle with dual-initiator systems.

[32] Ming Qiao, Jiazhong Xu, Bo You, Anlan Zou, and Deyou Guo. Axial temperature distribution optimization of mandrel for heated mandrel winding method. *Journal of Reinforced Plastics and Composites*, 31(9):631–638, 2012.

[33] Stephan Feldgoise, Michaelf Foley, David Martin, and James Bohan. The effect of microvoid content on composite shear strength. In *International SAMPE Technical Conference, 23 rd, Kiamesha Lake, NY*, pages 259–273, 1991.

[34] SR Ghiorse. Effect of void content on the mechanical properties of carbon/epoxy laminates. *SAMPE quarterly*, 24(2):54–59, 1993.

[35] Dominic Bertling, Robert Kaps, and Eyob Mulugeta. Analysis of dry-spot behavior in the pressure field of a liquid composite molding process. *CEAS Aeronautical Journal*, 7(4):577–585, 2016.

[36] Tomonaga Okabe, Yutaka Oya, Go Yamamoto, Junki Sato, Tsubasa Matsumiya, Ryosuke Matsuzaki, Shigeki Yashiro, and Shigeru Obayashi. Multi-objective

optimization for resin transfer molding process. *Composites Part A: Applied Science and Manufacturing*, 92:1–9, 2017.

[37] Dominic Bertling, Robert Kaps, and Eyob Mulugeta. Analysis of dry-spot behavior in the pressure field of a liquid composite molding process. *CEAS Aeronautical Journal*, 7(4):577–585, 2016.

[38] Mathieu Devillard, Kuang-Ting Hsiao, and Suresh G Advani. Flow sensing and control strategies to address race-tracking disturbances in resin transfer molding—part ii: automation and validation. *Composites Part A: Applied Science and Manufacturing*, 36(11):1581–1589, 2005.

[39] Masoud Bodaghi, Pavel Simacek, Nuno Correia, and Suresh G Advani. Experimental parametric study of flow-induced fiber washout during high-injection-pressure resin transfer molding. *Polymer Composites*, 41(3):1053–1065, 2020.

[40] D Nielsen and R Pitchumani. Intelligent model-based control of preform permeation in liquid composite molding processes, with online optimization. *Composites Part A: Applied Science and Manufacturing*, 32(12):1789–1803, 2001.

[41] Carlos González and Joaquín Fernández-León. A machine learning model to detect flow disturbances during manufacturing of composites by liquid moulding. *Journal of Composites Science*, 4(2):71, 2020.

[42] Simon Stieber, Niklas Schröter, Alexander Schiendorfer, and Alwin Hoffmann. Flowfrontnet: improving carbon composite manufacturing with cnns. 2020.

Chapter 2

Simulating resin injection in fibrous media using PINN

Abstract

In this chapter, we show the possibility of using PINN to simulate resin injection in fibrous media. The problem in hand is a two-phase flow (resin-air) in porous media; the macroscopic Darcy's law is used as an approximation to the momentum equation. The volume of fluids (VOF) technique is used to track the flow front between the two phases. It will be shown through some examples that having fixed collocation points lead to high errors in predicting the location of the flow front; thus an adaptive technique to enrich the collocation points based on the residuals is developed and tested which resolves the issue. The developed PINN framework along with adaptivity is tested on 1D and 2D problems.

Contents

2.1 Introduction

Modeling the process of resin injection in fibrous media is essential before the real process which is useful for process design such as deciding on the gate location or injection times. Moreover, it can help make predictions of the filling time and the possibility of defects. This is widely done using mesh-based techniques such as finite elements [1; 2], finite volumes [3; 4] and mixed finite element-volume formulations [5; 6].

The problem of resin injection in fibrous media involves a moving discontinuity, an interface between two fluids (resin and air), usually referred to as flow front. To capture the flow front movement across the computational grid, mesh-based techniques are coupled with methods such as level-set [7], the volume of fluids [8], or phase-field method [9]. Moreover, to accurately predict the location of the flow front, a very fine mesh is required which is computationally expensive. Another option is to adapt the mesh every few time steps (to have a denser mesh near the discontinuity) [10]. This option is cheaper than using one fine mesh, however, it is still costly to perform interpolation (transfer the data from one mesh to another) and less effective in 3D problems.

On the other hand, meshless and particle-based methods [11] have been proven to provide more natural ways of tracking the flow front. Smoothed-particle hydrodynamics can be regarded as the most commonly used meshless method which has great advantages in terms of ease of implementation, parallelization, and computational cost [12]. However, meshless methods in general are generally regarded as having a lower accuracy and reduced stability when compared to grid-based methods [13; 14].

Lately, Physics-informed neural networks (PINN) has gained widespread attention to solve problems represented by partial differential equations. That is due to the ease of implementation attributed to the use of automatic differentiation and that the technique is meshless. Moreover, the use of neural networks as the approximation space is a powerful choice due to their great approximation capabilities. Also, performing adaptivity by adding more collocation points in interesting regions is quite a straightforward and cheap task. PINN has been successfully used to solve problems in solid mechanics [15], fluid mechanics [16; 17], subsurface flow analysis [18], magnetic problems [19], and many others [20–27]. PINN showed great potential and success in these applications; for that, PINN is assessed in this chapter to solve resin injection in fibrous porous media.

The main objective of this chapter is to assess and develop a framework based PINN to be used as a meshless solver of resin injection in fibrous porous media (mold filling problem). In the process, a cheap (almost cost-free) adaptivity algorithm is developed and tested for 1D and 2D mold filling examples.

This chapter starts with the governing equations used to model two-phase flow in porous media. Afterwards, the PINN framework to solve the problem is detailed; the new adaptivity technique is introduced afterwards. Two classical examples (1D and

central injection) are addressed and the results are compared with existing analytical solutions. The chapter ends with a discussion and conclusion.

2.2 Two-phase flow in porous media model

To start with, figure 2.1 shows a generic geometry that can represent a mold domain Ω. The boundary of the domain is represented with the symbol Γ. The boundary includes an injection Γ_{in} and an outlet port Γ_{out}. The rest of the boundary is an impermeable wall Γ_{wall} where fluids cannot cross. At $t = 0$, the domain is completely filled with air with viscosity μ_a, then resin is injected under pressure $p_{in}(t)$ or controlled flow rate (velocity) $v_{in}(t)$ displacing the air outside the domain through Γ_{out} at $p_{out}(t)$. An interface (discontinuity) between the two fluid phases will always exist throughout the mold filling process.

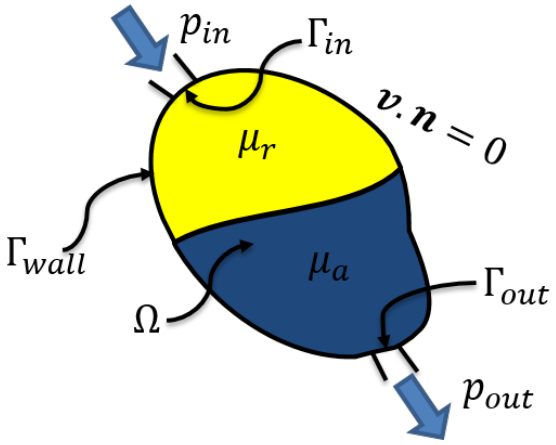

Figure 2.1: Generic mold geometry at time t showing the two fluid phases and the boundary conditions.

The flow in porous media is modeled using Darcy's law which is an approximation to the momentum equation. Darcy's law is written as:

$$\mathbf{v} = -\frac{1}{\mu}\mathbf{K} \cdot \nabla p \tag{2.1}$$

where $\mathbf{v}$ is the volume average Darcy's velocity, $\mathbf{K}$ the permeability tensor which is a property of the porous media measuring how easily the fluid can flow through the porous media, μ the dynamic viscosity, and ∇p the pressure gradient.

Both fluids are assumed to be incompressible, therefore, the mass conservation equation reduces to

$$\nabla \cdot \mathbf{v} = 0 \tag{2.2}$$

Inlet and outlet pressure boundary conditions are assigned as follows:

$$p(\mathbf{x}_{inlet}, t) = p_{in}(t) \text{ on } \Gamma_{in} \tag{2.3}$$
$$p(\mathbf{x}_{outlet}, t) = p_{out}(t) \text{ on } \Gamma_{in} \tag{2.4}$$

Inlet velocity can be assigned instead of the inlet pressure depending on the injection technique used

$$\mathbf{v}(\mathbf{x}_{inlet}, t) = v_{in}(t) \text{ on } \Gamma_{in} \tag{2.5}$$

Impermeable boundaries are characterized by zero normal velocity, thus

$$\mathbf{v} \cdot \mathbf{n} = 0 \quad \text{(Impermeable wall)} \tag{2.6}$$

To track the interface between the two fluid phases, the volume of fluid (VOF) method is used [28; 29]. It is based on defining a fraction function c, which is a scalar function that takes a value of 1 in the domain where resin exists, zero for the air, and values ranging from 0 to 1 near the interface between the phases. The more accurate the numerical method is, the smaller the interface region will be leading to a discontinuous fraction function in the ideal case. Using this definition, the viscosity μ in equation 2.1 can be rewritten as:

$$\mu = c\mu_r + (1 - c)\mu_a \tag{2.7}$$

where μ_r and μ_a are the viscosities of the resin and air, respectively. c evolves with time according to the following advection equation

$$c_t + \frac{1}{\phi}\mathbf{v} \cdot \nabla c = 0 \tag{2.8}$$

where c_t is the time derivative of the fraction function c and ϕ is the porosity of the porous media.

Initial and boundary conditions need to be defined for the VOF advection equation. The initial condition is written as:

$$c(\mathbf{x}, t = 0) = c_0(\mathbf{x})\,. \tag{2.9}$$

The mold is assumed to be completely filled with air at $t = 0$; thus, $c_0 = 0$ for all the upcoming examples.

Inlet flow also requires the assignment of boundary conditions for c:

$$c(\mathbf{x}_{inlet}, t) = 1 \quad \text{(Inlet)} \tag{2.10}$$

2.3 PINN structure

To solve this problem, we need, first, to approximate the pressure, fraction function, and velocity using feedforward neural networks. Several architectures can be used for this problem. One option is to use one single neural network to approximate all

the fields. A second option is to use distinct networks for the pressure and fraction function fields and one network for the different velocity components. Another is to use distinct networks for all the fields including distinct networks for each velocity component. We choose to use the third option since it was found by [30] that using distinct networks for the different fields of interest facilitates the optimization process. The PINN structure for a general two-phase flow in porous media problem is summarized in figure 2.2. Each of these networks has space $\mathbf{x}$ and time t inputs. The outputs of these neural networks ($\mathbf{v}$, p and c) are differentiated with respect to the inputs, using automatic differentiation [31], forming the residuals of the three differential equations (2.1), (2.2) and (2.8).

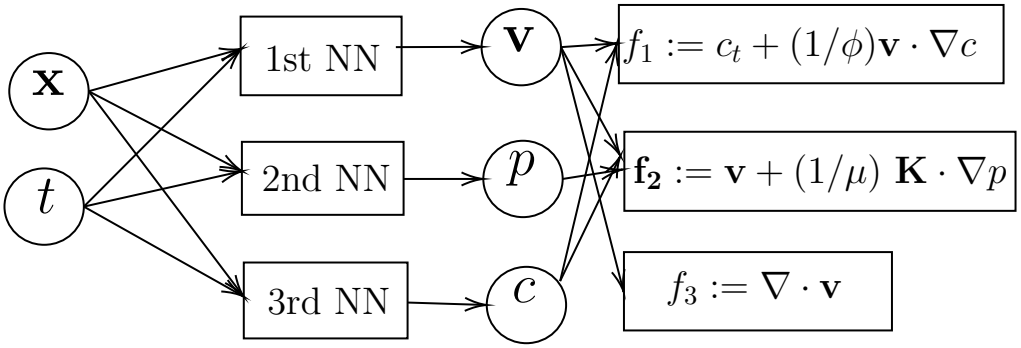

Figure 2.2: PINN structure for a general transient two-phase flow in porous media problem. The structure is composed of three neural networks, where each one has inputs of space and time. The networks output the velocity, pressure, and fraction function which are then used to form the different PDE residuals.

The loss function can, then, be defined as follows:

$$Loss = \lambda_v\ loss_v + \lambda_c\ loss_c + \lambda_p\ loss_p + \lambda_1\ loss_{f_1} + \lambda_2\ loss_{f_2} + \lambda_3\ loss_{f_3}, \quad (2.11)$$

where

$$loss_v = \frac{1}{N_v}\sum_{i=1}^{N_v} r_v^2(\mathbf{x_v^i}, t_v^i) = \frac{1}{N_v}\sum_{i=1}^{N_v} ||\mathbf{v}(\mathbf{x_v^i}, t_v^i) \cdot \mathbf{n}(\mathbf{x_v^i}, t_v^i)||^2, \quad (2.12)$$

$$loss_c = \frac{1}{N_c}\sum_{i=1}^{N_c} r_c^2(\mathbf{x_c^i}, t_c^i) = \frac{1}{N_c}\sum_{i=1}^{N_c} ||c(\mathbf{x_c^i}, t_c^i) - c_b^i||^2, \quad (2.13)$$

$$loss_p = \frac{1}{N_p}\sum_{i=1}^{N_p} r_p^2(\mathbf{x_p^i}, t_p^i) = \frac{1}{N_p}\sum_{i=1}^{N_p} ||p(\mathbf{x_p^i}, t_p^i) - p_b^i||^2, \quad (2.14)$$

$$loss_{f1} = \frac{1}{N_{f1}}\sum_{i=1}^{N_{f1}} ||f_1(\mathbf{x_{f1}^i}, t_{f1}^i)||^2 = \frac{1}{N_{f1}}\sum_{i=1}^{N_{f1}} ||c_t + (1/\phi)\mathbf{v} \cdot \nabla c||^2_{(\mathbf{x_{f1}^i}, t_{f1}^i)}, \quad (2.15)$$

$$loss_{f2} = \frac{1}{N_{f2}}\sum_{i=1}^{N_{f2}} ||f_2(\mathbf{x_{f2}^i}, t_{f2}^i)||^2 = \frac{1}{N_{f2}}\sum_{i=1}^{N_{f2}} ||\mathbf{v} + \frac{1}{\mu}\mathbf{K} \cdot \nabla p||^2_{(\mathbf{x_{f2}^i}, t_{f2}^i)}, \quad (2.16)$$

$$loss_{f3} = \frac{1}{N_{f3}} \sum_{i=1}^{N_{f3}} ||f_3(\mathbf{x_{f3}^i}, t_{f3}^i)||^2 = \frac{1}{N_{f3}} \sum_{i=1}^{N_{f3}} ||\nabla \cdot \mathbf{v}||^2_{(\mathbf{x_{f3}^i}, t_{f3}^i)}, \tag{2.17}$$

and $\{\mathbf{x_v^i}, t_v^i, v_b^i\}_{i=1}^{N_v}$, $\{\mathbf{x_c^i}, t_c^i, c_b^i\}_{i=1}^{N_c}$ and $\{\mathbf{x_p^i}, t_p^i, p_b^i\}_{i=1}^{N_p}$ are the points where the initial/boundary conditions are defined for $\mathbf{v}$, c and p, respectively. While $\{\mathbf{x_{f1}^i}, t_{f1}^i\}_{i=1}^{N_{f1}}$, $\{\mathbf{x_{f2}^i}, t_{f2}^i\}_{i=1}^{N_{f2}}$ and $\{\mathbf{x_{f3}^i}, t_{f3}^i\}_{i=1}^{N_{f3}}$ are the collocation points in space and time for the three residuals, respectively, where the physics are enforced, and λ_i are the weights of each term in the loss function. The λ_i weights are important to make each contribution to the loss function has comparable magnitudes thus, helping the optimization process. Finally, a solution for the fields, p, $\mathbf{v}$, and c, is obtained by minimizing the loss function with respect to the neural networks' parameters.

Note

It should be noted that choosing the λ_i weights is an open research topic in the machine learning community and PINN as well. In this study, non-dimensionalization is performed before the training process, however, in some cases, manual tuning of these weights was done.

2.4 Residual-based adaptivity in PINN

Since, the filling problem deals with a moving discontinuity (front between the two fluid phases), having a fixed number of collocation points in the spatio-temporal domain might lead to inaccurate solutions and smearing of the discontinuity. Thus, we chose to adapt the number and location of the collocation points during the training process; this can be seen as similar to the mesh refinement in mesh-based techniques, however the degrees of freedom in PINN (weights and biases of the neural networks) do not change.

Mesh refinement is a basic idea in classical numerical techniques such as finite element (FEM) and finite volume (FVM) methods [32]. There are three basic techniques to mesh adaptation: h-adaptivity [33], r-adaptivity [34] and p-adaptivity [35]. H-adaptivity adds more nodes, thus increasing the degrees of freedom and the mesh connectivity. While r-adaptivity keeps the same number of nodes and degrees of freedom however the nodes are relocated while keeping the same connectivity. Finally, p-adaptivity increases the polynomial degrees of elements while keeping the mesh fixed. Other adaptivity methods exist that combines some of the three basic techniques together such as: hp-adaptivity [36] and rh-adaptivity [35].

There are mainly three drivers for mesh adaptation: error [37], PDE residual [38] and solution features [39].

Error-based adaptation adds more degrees of freedom where the solution error is high. This technique ignores the fact that the error is transported in the domain. Therefore, adapting where the error is high might ignore the region of the error

source itself, where adaption is more useful [40].

Residual-based adaptation refines the mesh where the discretized PDE residual is high. The residual can be seen as the source of error in the solution [41]. Thus, refining where the residual is high is seen as a way of refining where the error source is. Therefore, this technique usually performs better than error-based adaptation.

Solution-based adaptation utilizes the solution features such as gradients or discontinuities for adaptation. The philosophy behind this technique is that by using more points in these locations, these features can be resolved, thus, leading to improving the overall accuracy of the solution. However, if multiple features are present in the same problem, the adaptation results in over-refining some features while others are ignored. An example of this adaptivity failure can be found in [42].

The adaptation process is usually computationally expensive since certain requirements have to be satisfied and the mesh connectivity needs to be updated. Moreover, parallelization becomes complex for unstructured grids. However, in the case of PINN, changing the collocation points locations or adding more points are cheap processes. The main reason is that PINN is a meshless method, thus, there are no specific element topologies to respect or mesh connectivity to update. Moreover, the approximation of the derivatives is independent of the collocation point position. Therefore, there is no discretization error resulting from the distribution of the collocation points. The only thing to do is identify the location where more points are needed.

In this section, we show the development of a residual-based algorithm by enriching the locations where the residual is high with more collocation points. We build the algorithm based on the work of [43] in which the authors developed the residual-based adaptive refinement method (RAR). In their work, the authors used a dense set of randomly drawn points in the space-time domain, where residuals are evaluated. Points corresponding to the largest residual values are then added to the training set of collocation points. The progressive refinement of the training set allows for residual control. However, when residual is showing high values in very narrow regions, this sampling strategy tends to produce excessively clustered points ignoring other solution features if existing and leading to unnecessary over-refinement. This behavior appears to be related to solutions exhibiting moving sharp fronts or discontinuities, as in our model. An example of that is seen in figure 2.3 where the added points are focused in a very small region.

Note

It should be noted that the dense set of collocation points is only used for obtaining the residuals (forward pass). This process is only done few times during the full training process; thus, it is computationally cheap to add extra points in the training set.

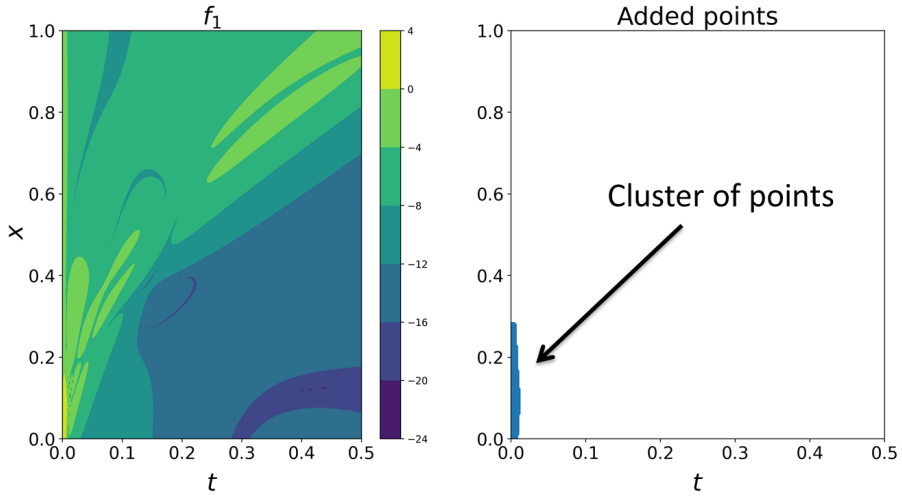

Figure 2.3: Left: log of the absolute residual field f_1, right: the chosen points according to RAR.

To avoid this collocation point clustering that might lead to over-fitting of the model, we designed a probability density function based on the residual field to control the spread of the added points. The density function is, then, used to draw points from the dense set and these points are added to the training set. By doing this, collocation points will be more evenly spread in the domain; more points will be added where the probability is high (high residual) and fewer points where the probability is lower (low residual). Figure 2.4 explains the procedure used for the adaptivity technique developed.

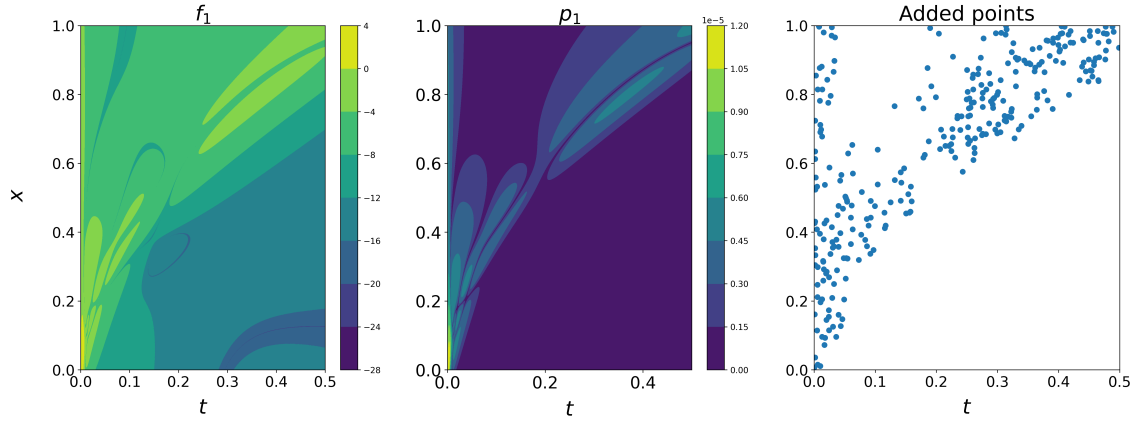

Figure 2.4: Left: log of the absolute residual field f_1, middle: the probability density function built from the residual, right: the chosen points drawn from the density function.

The algorithm is extended for coupled differential equations so that each PDE residual will have different collocation points. Moreover, data points are also enriched using a similar strategy to better capture the initial/boundary conditions. The algorithm is as follows:

Algorithm 1: Residual-based adaptivity

Inputs: Number of adaptivity steps M, number of iterations n, tolerances ϵ_i, ϵ_v, ϵ_c and ϵ_p;

while $m < M$ & $(\mu_i > \epsilon_i \;||\; \mu_v > \epsilon_v \;||\; \mu_c > \epsilon_c \;||\; \mu_p > \epsilon_p)$ **do**

- Calculate f_i (3 PDE residuals), r_v, r_c and r_p using dense sets;
- Build the probability functions p_i, p_v, p_c and p_p using dense sets Eq: 2.18;
- Draw points from the dense sets using the probabilities and add it to the training sets;
- Use minimization algorithm for n fixed iterations;
- Calculate the residuals' mean using the dense sets;

$$\mu_i = \frac{1}{N_i} \sum |f_i|,$$
$$\mu_v = \frac{1}{N_v} \sum |\mathbf{v}(\mathbf{x_v^i}, t_v^i) \cdot \mathbf{n}(\mathbf{x_v^i}, t_v^i)|,$$
$$\mu_c = \frac{1}{N_c} \sum |c(x_c^i, t_c^i) - c_b^i|,$$
$$\mu_p = \frac{1}{N_p} \sum |p(x_p^i, t_p^i) - p_b^i|,$$

end

The probability functions used have the form of

$$p(\mathbf{X}) = \frac{\max(\log |r(\mathbf{X})/\epsilon|, 0)}{\int_\Omega \max(\log |r(\mathbf{X})/\epsilon|, 0)\, d\mathbf{X}}, \tag{2.18}$$

where $\mathbf{X}$ is the random vector $[x, t]^T$, r the considered residual, Ω the spatio-temporal domain and ϵ a small tolerance to filter small residual values. In practice, the choice of the value of ϵ is chosen to control the spread of the point distributions. The function is designed in a way to ensure that its integral over the space-time domain is 1, hence, the presence of the term in the denominator which is calculated using Monte Carlo integration over the dense set of points.

2.5 Numerical examples

In this section, we provide numerical examples to assess the ability of PINN to simulate the resin injection process and also to assess the developed adaptivity algorithm. Two classical examples are provided: a 1D injection and a 2D central injection with an orthotropic permeability tensor.

The system of equations to be solved is initially non-dimensionalized so that the different terms of the loss function will have comparable magnitudes, thus, the training process will be simpler.

2.5.1 One dimensional injection

Physical model and scenarios

We consider a one-dimensional problem shown in figure 2.5. At $t = 0$, the domain is completely filled with air. The inlet is located at the left end where resin is being injected at constant pressure p_{in}. The outlet is located at the right end, at a distance l from the inlet, where the pressure is kept constant at p_{out}, which usually corresponds to atmospheric pressure. The permeability of the domain k is assumed to be homogeneous and constant in time.

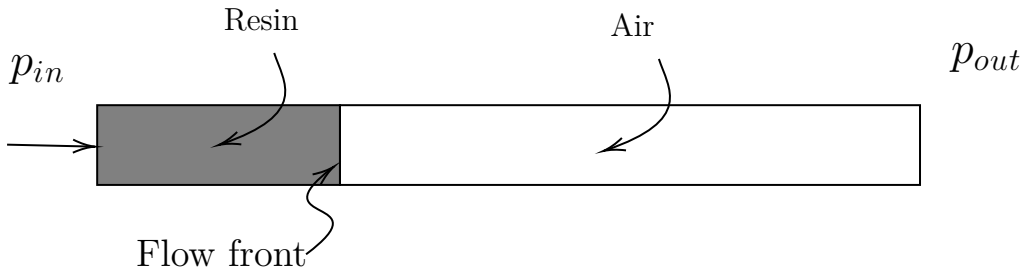

Figure 2.5: One-dimensional porous domain (filling problem)

Since the domain is initially filled with air, the fraction function c is 0 at $t = 0$ ($c(x, 0) = 0$). Moreover, at $x = 0$ the resin is being injected continuously over time, thus, $c(0, t) = 1$. These two conditions create a discontinuity at $t = x = 0$; this discontinuity is advected with the flow velocity.

To simplify the process of minimizing the combined loss function, the problem is nondimensionalized. This allowed us to weigh the different terms in the loss function in Eq 3.1 similarly. Therefore, $\lambda_c = \lambda_p = \lambda_1 = \lambda_2 = \lambda_3 = 1$. It should be noted that there are no conditions on the velocity field, thus the term $loss_v$ is not included in the full loss function definition. The parameters used in the nondimensionalized 1D problem are shown in table 2.1.

Table 2.1: Parameters used for the nondimensionalized 1D two-phase flow.

Parameter	Symbol	Value
Length	l	1
Full time	T	0.5
Permeability	k	1
Resin viscosity	μ_r	1
Air viscosity	μ_a	10^{-5}
Inlet pressure	p_{in}	1
Outlet pressure	p_{out}	0
Porosity	ϕ	1

In this example, three distinct neural networks are used to approximate the pressure, fraction function, and velocity fields. Each network has two inputs x and t and one output corresponding to the field being approximated. Each network has 5 hidden layers and 20 neurons. The hyperbolic tangent activation function is used in

all the hidden layers since it was found that it provides good results according to [18; 25; 30]. However, for the output layer, the sigmoid activation function is used for the pressure and fraction function networks since their values go from 0 to 1. While the linear activation function is used for the output layer of the velocity network.

The optimization strategy of using Adam optimizer followed by a second order optimize as BFGS is followed by [17; 43; 44]. The reason is that second-order methods like BFGS are prone to fall into local minima, therefore, Adam is firstly used to reach the zone of the global minimum, afterwards, BFGS is used to reach the minimum more easily since it iteratively builds an approximation of the inverse Hessian matrix which is used to reach the minimum. The details of the neural network architecture used and optimization details are shown in table 2.2.

Table 2.2: Neural Network architecture and optimization method used for the nondimensionalized 1D two-phase flow.

Parameter	Value
no. of hidden layers	5
no. neurons per hidden layer	20
Activation function of hidden layers	tanh
Optimization	1000 Adam iteration then 500 BFGS
Adam learning rate	0.001

Three numerical experiments are performed and compared to assess the developed adaptivity technique. In the first experiment, a fixed number and location of collocation points are used (2500 points organized in 50×50 grid) for the whole training phase. For the second experiment, RAR technique is assessed starting with 1600 collocation points organized as 40×40 grid points during the Adam training phase. Afterwards, point enrichments are performed 3 times every during the BFGS minimization stage till the stopping criteria are satisfied (at 2500 points as well). The final experiment using the provided adaptivity algorithm starting with 1600 points organized as 40×40 grid points during the Adam training phase. Similar enrichments are done as in the RAR experiment. Table 2.3 summarizes the three scenarios. All cases took nearly 200 seconds to converge using a laptop with Intel core i7-6700HQ CPU @ 2.60 GHz 2.59 GHz with 8 Go RAM.

Table 2.3: Summary of the collocation points evolution in the 3 numerical experiments for the nondimensionalized 1D two-phase flow.

	Starting points	Adaptation	Final points
Fixed PINN	2500 (50 × 50 grid)	No	2500
PINN with RAR	1600 (40 × 40 grid)	3 enrichments	2500
New Adaptive PINN	1600 (40 × 40 grid)	3 enrichments	2500

For testing the generality of the solution, a fixed 1000 points randomly distributed over the domain are used to evaluate the loss function. This test set will provide a sense of the generalization error committed during the training phase. It should be noted that these points are only used for testing but are not used in the training phase.

Flow front position

The analytical solution for the flow front x_f as a function of time is obtained and is written as:

$$x_f = \frac{-\mu_a l + \sqrt{\mu_a^2 l^2 + 2(\mu_r - \mu_a)k(p_{in} - p_{out})t}}{\mu_r - \mu_a}. \tag{2.19}$$

The flow front position in the three numerical experiments is extracted as the 0.5 level set of the fraction function. The front positions from the numerical tests are plotted along with the analytical solution in figure 2.6.

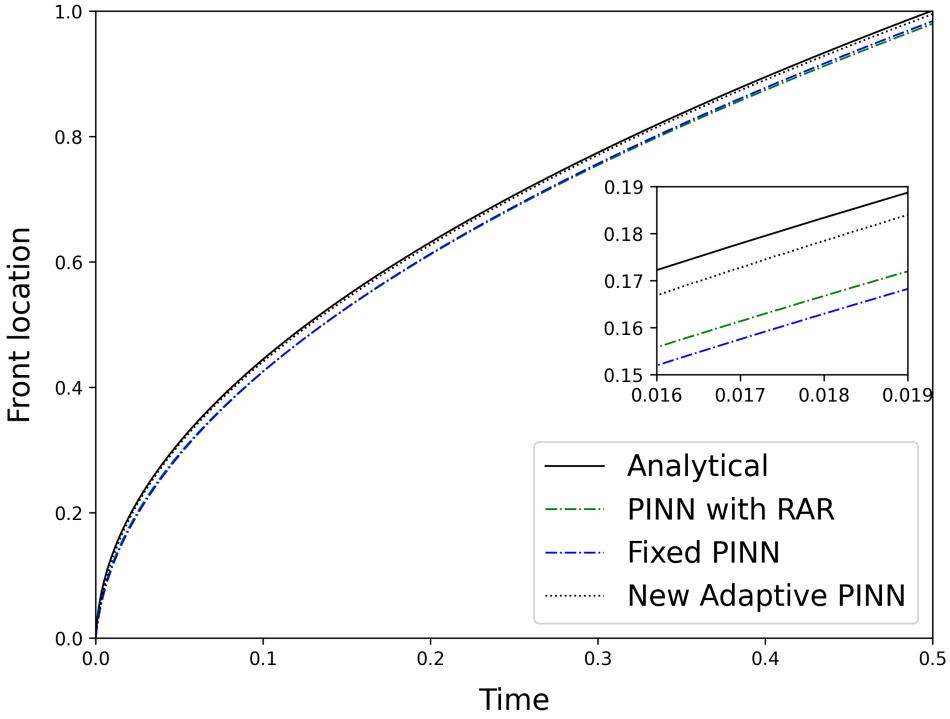

Figure 2.6: Front position with time for the fixed, RAR, and new adaptivity cases along with the analytical front position for the 1D filling example and a zoomed image to differentiate.

It can be seen from figure 2.6 that the proposed adaptivity technique provided the best results (closer to the analytical solution) among the three scenarios. PINN with RAR did not provide a significant improvement to using fixed points; that is probably due to the focusing of the enrichments in a small spatiotemporal region leading to harder optimization, and higher generalization error, thus leading to minimal improvement. However, the newly developed adaptivity method provided a greater improvement probably due to the widespread of the enrichment points leading to better generalization.

Pressure profiles

The pressure profiles as a function of x at different times are shown in figure 2.7 for the three scenarios and compared to the existing analytical solution. The analytical solution for pressure can be written as

$$p(x,t) = \begin{cases} \dfrac{-\mu_r(p_{in}-p_{out})}{(\mu_r-\mu_a)x_f(t)+\mu_a l}\, x + p_{in} & x < x_f(t) \\ \dfrac{-\mu_a(p_{in}-p_{out})}{(\mu_r-\mu_a)x_f(t)+\mu_a l}\, x + \dfrac{\mu_a(p_{in}-p_{out})}{(\mu_r-\mu_a)x_f(t)+\mu_a l}\, l + p_{out} & x \geq x_f(t) \end{cases} . \tag{2.20}$$

The new adaptive case provided a pressure solution closer to the analytical solution than both of the fixed PINN and PINN with RAR cases. PINN with RAR shows a prediction far from the analytical solution near $t = 0$.

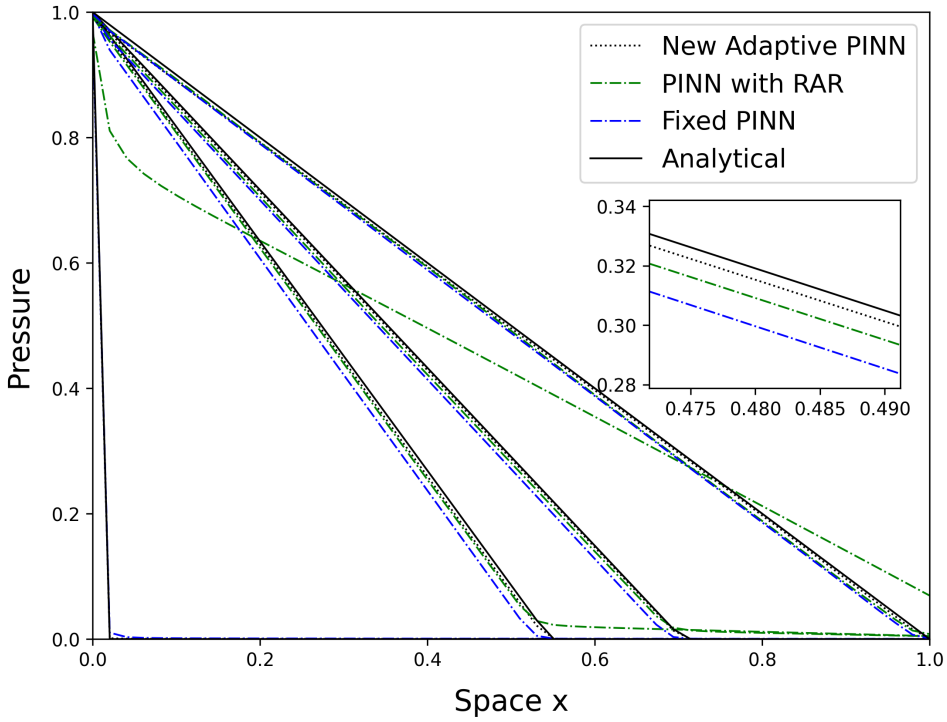

Figure 2.7: Pressure profiles at different times ($t = 0$, $t = 0.15$, $t = 0.25$ and $t = 0.5$) for the adaptive and fixed collocations cases for the 1D filling example.

As can be seen from figure , the pressure is initially almost zero in the whole domain since it is fully filled with air. However, at $x = 0$, the pressure should be 1 since the domain is being filled under constant pressure. This condition creates a discontinuity at $t = 0$. At further times, the pressure has a linear profile with a kink at the flow front.

Evolution of the collocation points distribution

The distribution of the collocation points for the different PDEs is shown in the case of new adaptive PINN in figure 2.8. The figure shows the evolution of these distributions at different stages of using the adaptivity algorithm.

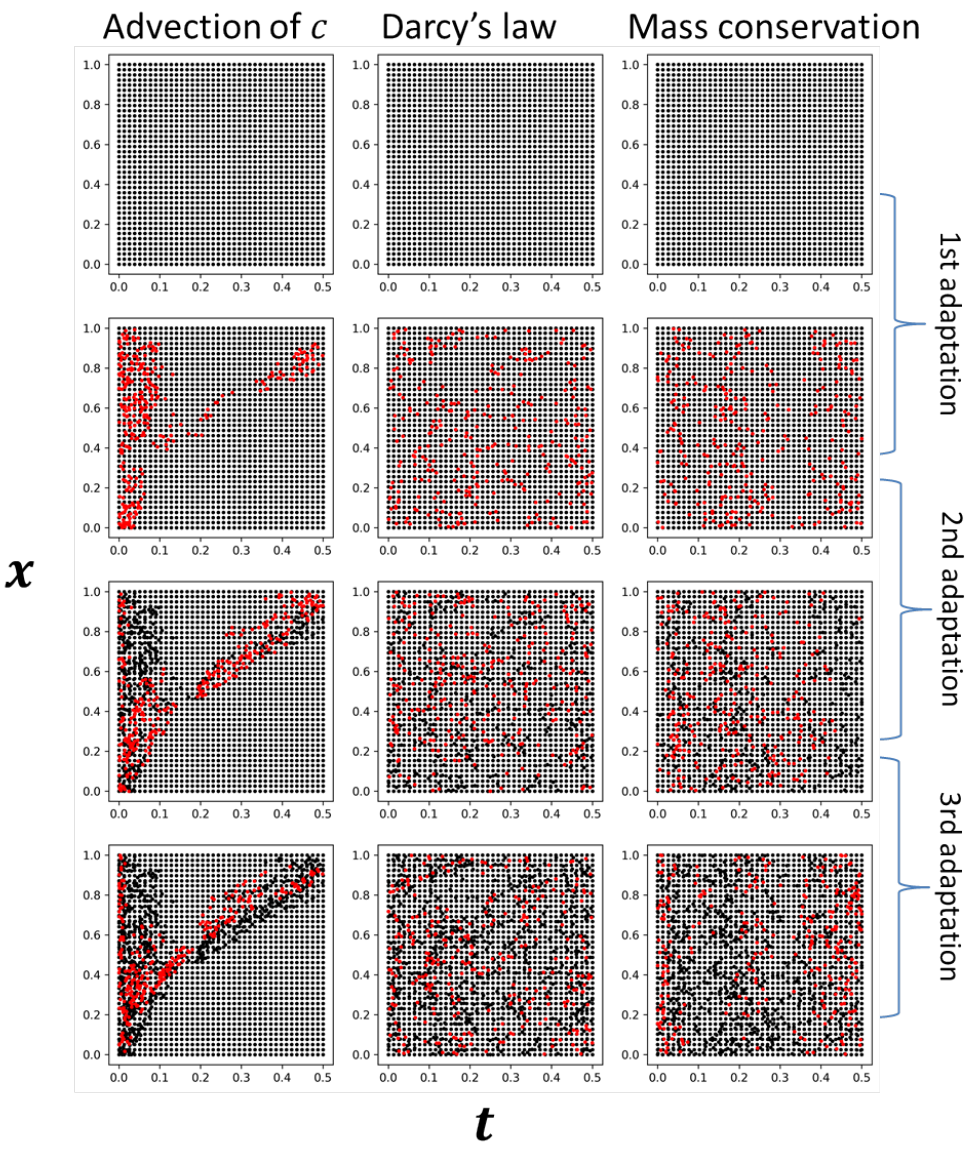

Figure 2.8: Evolution of collocation points with new adaptivity (from left to right: collocations for f_1, collocations for f_2 and collocations for f_3) for the 1D filling example. New collocation points after each adaptation step are shown in red.

From figure 2.8, it can be seen for the collocation points of f_1 that the points seem to be dense near the location of the front, where the residual is higher. This helped in capturing the interface location accurately. While for the other collocation points (f_2 and f_3), they were distributed almost randomly in the domain. That is because the residual field is spread all over the domain since there are no sharp solution features to capture.

Loss function and generalization

The loss function is compared in the three scenarios by plotting the loss vs. iteration graph for both cases. The loss using the training set is compared to that using the testing set for all cases as shown in figure 2.9. It should be noted that there

is a deviation between the training and testing loss in the fixed collocations case, meaning that the generalization error is high. This deviation is marginally decreased in the PINN with RAR case. For the new adaptive PINN case, the deviation significantly decreases meaning that less generalization error is committed using the newly-developed adaptive technique. From a deep learning perspective, using adaptive collocation points can be seen as a form of regularization of the neural network solution; adaptivity prevents overfitting, thus making the solution more accurate for unseen points (points not used in the training process).

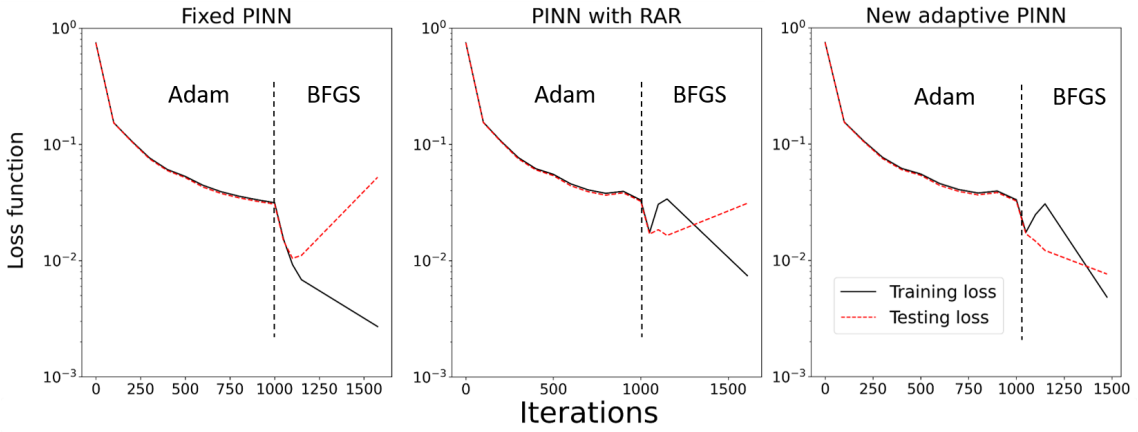

Figure 2.9: Loss vs. Iteration graphs for fixed PINN (left plot), new adaptive PINN (middle plot), and PINN with RAR (right plot) for the 1D filling example.

The different terms in the loss function are plotted for the new adaptive case against the number of iterations in figure 2.10. This is done to assure the convergence of all the terms.

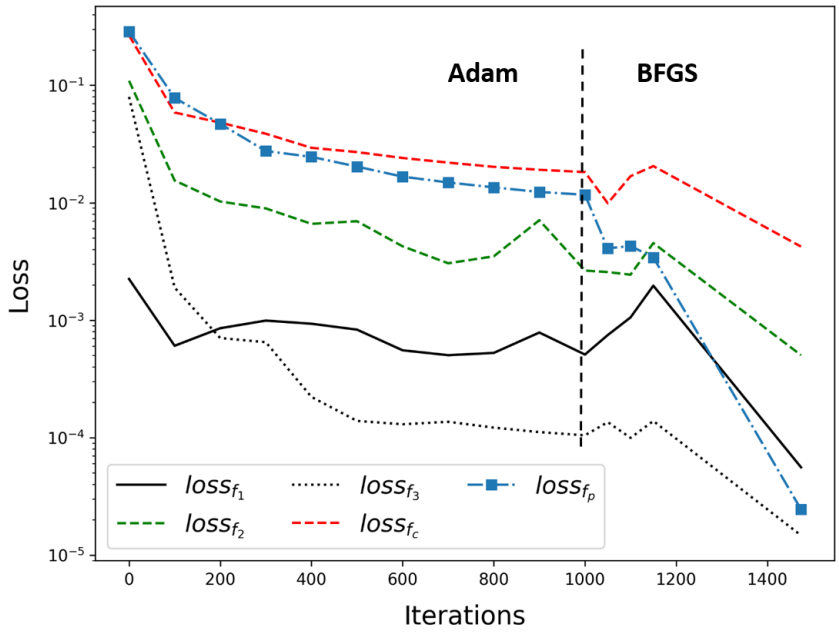

Figure 2.10: Different loss terms versus iterations for the new adaptivity case for the 1D filling example.

All the loss terms have more or less a similar trend in convergence. We can note that the loss corresponding to the initial and boundary conditions of the fraction function c has the highest values (harder convergence) that is due to the discontinuity in the initial/boundary condition values at $t = 0$.

2.5.2 Two dimensional central injection

Physical model and scenarios

The next example is a two-dimensional central injection problem. The domain is a square of a unity area with an elliptic injection port placed at the center with constant pressure ($p = 1$). The four outer sides are outlets where the pressure is set to zero. The problem domain is plotted in figure 2.11. The analytical solution to this problem exists in [45].

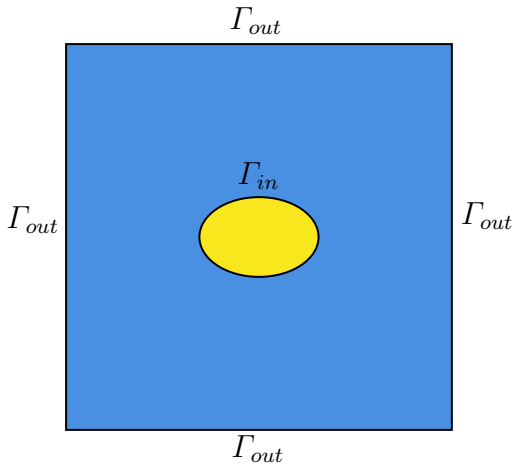

Figure 2.11: Domain of the 2D central injection problem. Γ_{out} is the outlet boundary where the pressure is set to 0, while Γ_{in} is the boundary of the inlet where pressure is set to 1.

The material properties are the same as in the first example (table 2.1) except for the permeability of the domain. The permeability is assumed to be homogeneous, constant in time, and orthotopic, where the permeability tensor can be written as

$$\mathbf{K} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.21}$$

All training parameters, that were used in the 1D filling example (section 2.5.1), are used for the 2D example except for the network architecture. In this case, 4 neural networks are used to approximate the fraction function, pressure and two velocity components; each network is composed of 5 hidden layers and 20 neurons. The 3 scenarios (Fixed PINN, PINN with RAR and new adaptive PINN) are compared in the 2D filling example, as well.

Flow front position and pressure fields

The fraction function and pressure fields are plotted at different times for the adaptive case to visualize the evolution of the flow front and pressure with time in figure 2.12.

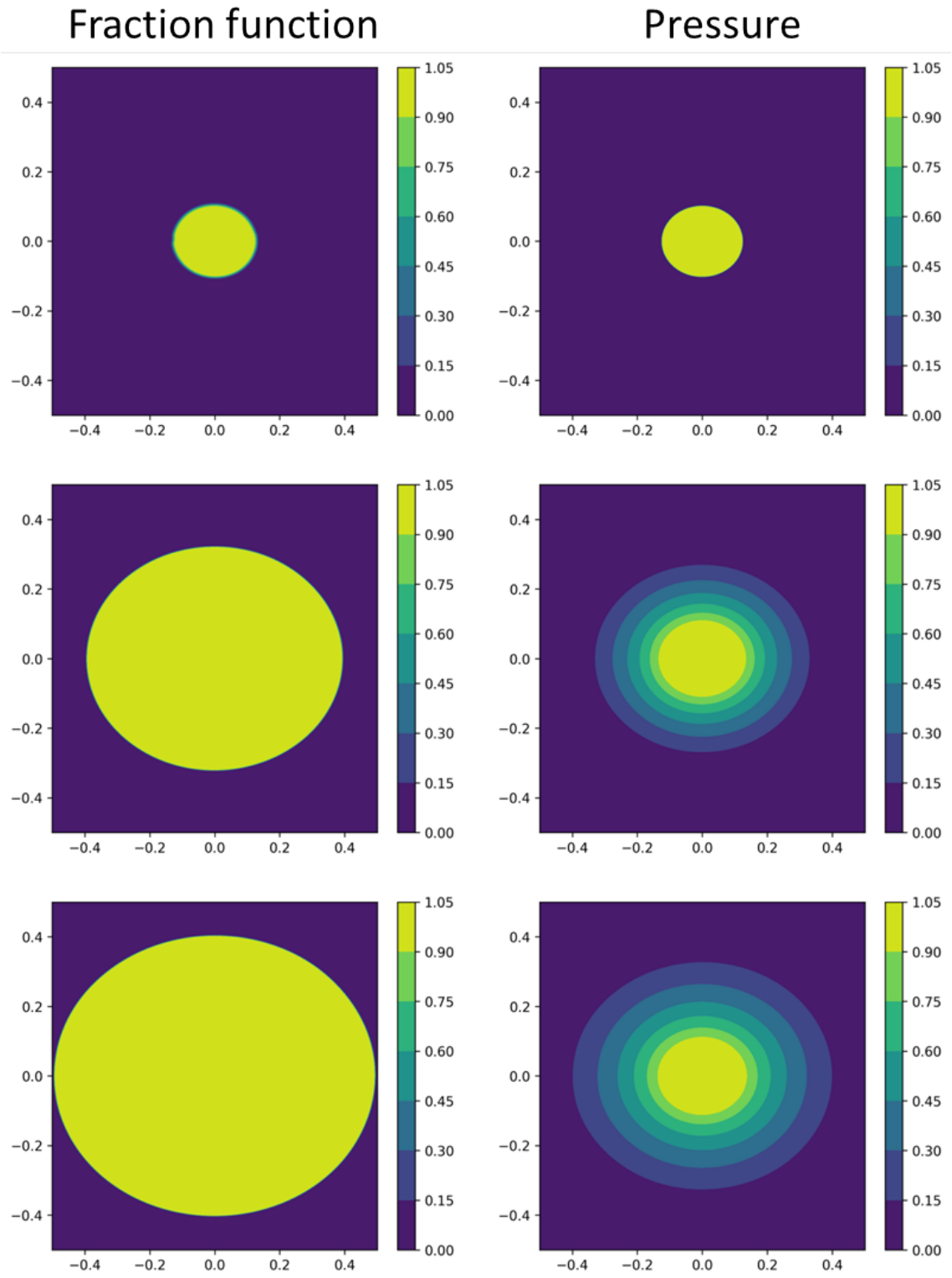

Figure 2.12: Left: Fraction function field. Right: pressure field. The plots are shown at times 0, 0.039 and 0.078 from top to bottom for the 2D filling example.

The positions of the flow front along the x and y directions are extracted as a function of time and plotted in figures 2.13 and 2.14. The results are shown for the 3 numerical experiments along with the existing analytical solution for comparison.

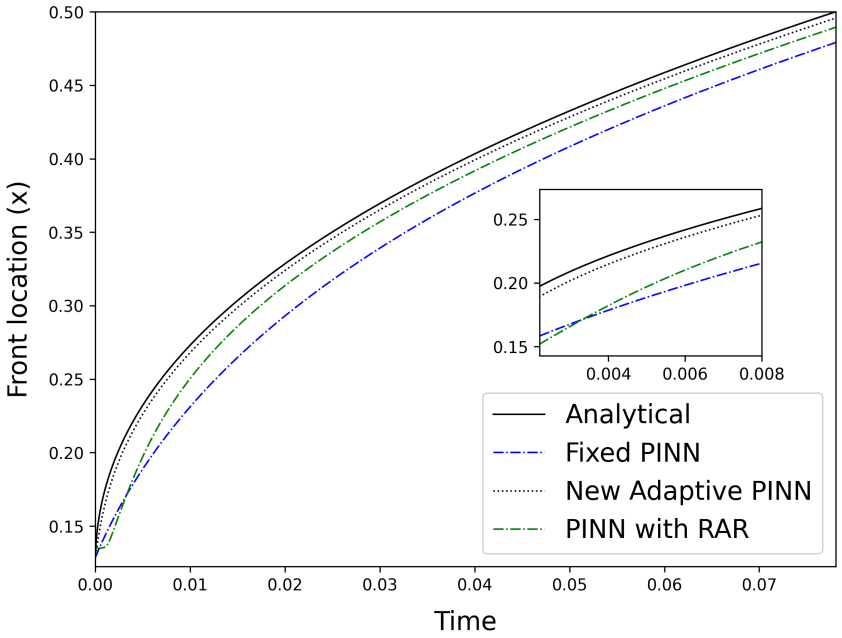

Figure 2.13: Front (fluid 1/fluid 2 interface) position in the x-direction with time for the fixed and adaptive cases along with analytical front position for the 2D filling example.

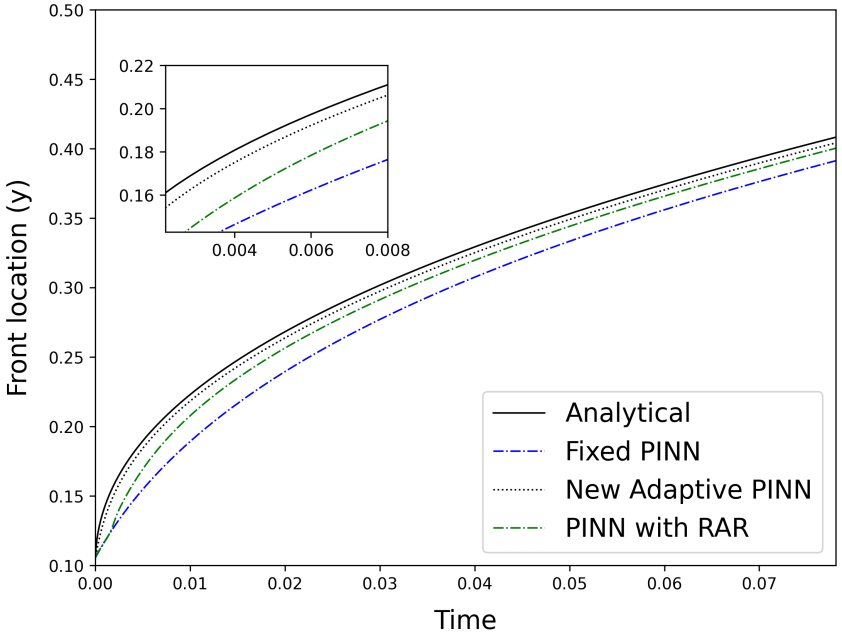

Figure 2.14: Front (fluid 1/fluid 2 interface) position in the y-direction with time for the fixed and adaptive cases along with analytical front position for the 2D filling example.

It can be seen that the new adaptive algorithm provided a better prediction (closer to the analytical solution) of the flow front position.

Loss function and generalization

The loss function is plotted for the 3 cases in figure 2.15. 10,000 points in the space-time domain are chosen randomly to assess the testing loss, while they are not used in the training.

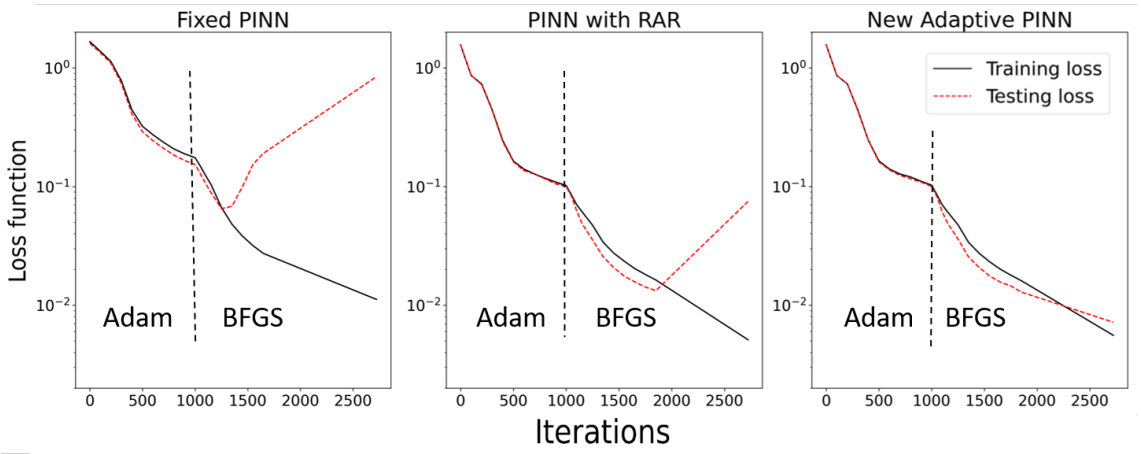

Figure 2.15: Loss vs. Iteration graphs for fixed PINN (left plot), PINN with RAR (middle plot), and new adaptive PINN (right plot).

Using the new adaptivity algorithm, the discrepancy between the training and testing loss is greatly reduced. That means that it offers a mean to reduce the generalization error thus provides better accuracy to the approximated fields.

2.6 Summary and conclusion

In this chapter, we have seen:

- the system of equations used to model the resin injection in porous media,
- the used PINN framework to solve the problem,
- the new adaptivity technique that was developed to better capture the flow front,
- and two numerical examples (1D and 2D) to validate the developed techniques.

It can be concluded from this chapter, that PINN has the ability to simulate the process of resin injection in porous media. Performing adaptivity with PINN improves the accuracy of the results while being computationally cheap as compared to performing adaptivity in mesh-based techniques. The main originalities in this chapter are:

- building a 1D and 2D solver for mold filling in porous media.
- developing a general residual-based adaptive algorithm.

Transition

Online control or process design requires changing several parameters such as the injection location or material parameters. This leads to the need of doing many simulations in which the number of simulations can easily grow exponentially with the number of parameters to change. To avoid that, metamodels are built offline and can be used really quickly online without the need for retraining. Building metamodels using PINN will be the subject of the next chapter.

2.7 Bibliography

[1] F Trochu, R Gauvin, and D-M Gao. Numerical analysis of the resin transfer molding process by the finite element method. *Advances in Polymer Technology: Journal of the Polymer Processing Institute*, 12(4):329–342, 1993.

[2] Mark Lin, H Thomas Hahn, and Hoon Huh. A finite element simulation of resin transfer molding based on partial nodal saturation and implicit time integration. *Composites Part A: Applied Science and Manufacturing*, 29(5-6):541–550, 1998.

[3] Julian Seuffert, Luise Kärger, and Frank Henning. Simulating mold filling in compression resin transfer molding (crtm) using a three-dimensional finite-volume formulation. *Journal of Composites Science*, 2(2):23, 2018.

[4] Jeferson Avila Souza, Luiz Alberto Oliveira Rocha, Sandro Campos Amico, and José Viriato Coelho Vargas. A numerical investigation of the resin flow front tracking applied to the rtm process. *Materials Research*, 14:345–354, 2011.

[5] MV Bruschke and Suresh G Advani. A finite element/control volume approach to mold filling in anisotropic porous media. *Polymer composites*, 11(6):398–405, 1990.

[6] Jamal Samir, Jamal Echaabi, and Mohamed Hattabi. Control volume finite element methods for flow in porous media: Resin transfer molding. *Finite Element Analysis–Applications in Mechanical Engineering*, 2012.

[7] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational physics*, 114(1):146–159, 1994.

[8] Cyril W Hirt and Billy D Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.

[9] Long-Qing Chen. Phase-field models for microstructure evolution. *Annual review of materials research*, 32(1):113–140, 2002.

[10] Eric Béchet, E Ruiz, and François Trochu. Adaptive mesh generation for mould filling problems in resin transfer moulding. *Composites Part A: Applied Science and Manufacturing*, 34(9):813–834, 2003.

[11] Nazia Talat, Boštjan Mavrič, Grega Belšak, Vanja Hatić, Saša Bajt, and Božidar Šarler. Development of meshless phase field method for two-phase flow. *International Journal of Multiphase Flow*, 108:169–180, 2018.

[12] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[13] Zhi-Bin Wang, Rong Chen, Hong Wang, Qiang Liao, Xun Zhu, and Shu-Zhe Li. An overview of smoothed particle hydrodynamics for simulating multiphase flow. *Applied Mathematical Modelling*, 40(23-24):9625–9655, 2016.

[14] M Sawley, P Cleary, and Joseph Ha. Modelling of flow in porous media and resin transfer moulding using smoothed particle hydrodynamics. 1999.

[15] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A deep learning framework for solution and discovery in solid mechanics. *arXiv preprint arXiv:2003.02751*, 2020.

[16] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[17] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[18] Alexandre M Tartakovsky, C Ortiz Marrero, Paris Perdikaris, Guzel D Tartakovsky, and David Barajas-Solano. Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems. *Water Resources Research*, 56(5):e2019WR026731, 2020.

[19] Alexander Kovacs, Lukas Exl, Alexander Kornell, Johann Fischbacher, Markus Hovorka, Markus Gusenbauer, Leoni Breth, Harald Oezelt, Dirk Praetorius, Dieter Suess, et al. Magnetostatics and micromagnetics with physics informed neural networks. *Journal of Magnetism and Magnetic Materials*, 548:168951, 2022.

[20] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[21] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[22] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[23] Khemraj Shukla, Patricio Clark Di Leoni, James Blackshire, Daniel Sparkman, and George Em Karniadakis. Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks. *Journal of Nondestructive Evaluation*, 39(3):1–20, 2020.

[24] Qiming Zhu, Zeliang Liu, and Jinhui Yan. Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks. *Computational Mechanics*, 67(2):619–635, 2021.

[25] Sina Amini Niaki, Ehsan Haghighat, Trevor Campbell, Anoush Poursartip, and Reza Vaziri. Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. *Computer Methods in Applied Mechanics and Engineering*, 384:113959, 2021.

[26] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[27] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[28] Cyril W Hirt and Billy D Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.

[29] CR Swaminathan and VR Voller. A time-implicit filling algorithm. *Applied Mathematical Modelling*, 18(2):101–108, 1994.

[30] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 2021.

[31] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

[32] Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.

[33] Pedro Díez and Antonio Huerta. A unified approach to remeshing strategies for finite element h-adaptivity. *Computer Methods in Applied Mechanics and Engineering*, 176(1-4):215–229, 1999.

[34] Harm Askes and Antonio Rodríguez-Ferran. A combined rh-adaptive scheme based on domain subdivision. formulation and linear examples. *International Journal for numerical methods in engineering*, 51(3):253–273, 2001.

[35] Harm Askes and Antonio Rodríguez-Ferran. A combined rh-adaptive scheme based on domain subdivision. formulation and linear examples. *International Journal for numerical methods in engineering*, 51(3):253–273, 2001.

[36] Ivo Babuška and Manil Suri. The p and h-p versions of the finite element method, basic principles and properties. *SIAM review*, 36(4):578–632, 1994.

[37] Xu Dong Zhang, J-Y Trépanier, and Ricardo Camarero. A posteriori error estimation for finite-volume solutions of hyperbolic conservation laws. *Computer methods in applied mechanics and engineering*, 185(1):1–19, 2000.

[38] D Lee and YM Tsuei. A formula for estimation of truncation errors of convection terms in a curvilinear coordinate system. *Journal of Computational Physics*, 98(1):90–100, 1992.

[39] D Scott McRae. r-refinement grid adaptation algorithms and issues. *Computer Methods in Applied Mechanics and Engineering*, 189(4):1161–1182, 2000.

[40] Aniruddha Choudhary and Christopher Roy. Efficient residual-based mesh adaptation for 1d and 2d cfd applications. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 214, 2011.

[41] Christopher Roy. Strategies for driving mesh adaptation in cfd. In *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, page 1302, 2009.

[42] Richard P Dwight. Heuristic a posteriori estimation of error due to dissipation in finite volume schemes and application to mesh adaptation. *Journal of Computational Physics*, 227(5):2845–2863, 2008.

[43] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[44] Teeratorn Kadeethum, Thomas M Jørgensen, and Hamidreza M Nick. Physics-informed neural networks for solving nonlinear diffusivity and biot's equations. *PloS one*, 15(5):e0232683, 2020.

[45] JR Weitzenböck, RA Shenoi, and PA Wilson. Radial flow permeability measurement. part a: Theory. *Composites Part A: Applied Science and Manufacturing*, 30(6):781–796, 1999.

Chapter 3

Building Metamodels with PINN

Abstract

In this chapter, we show the ability of PINN to build metamodels related to resin injection in a fibrous media process. The method involves adding new inputs to the network representing parameters of interest. Collocation points are, then, sampled in space-time-parameters space. The resulting metamodel can be used for quick predictions, process design, or control.

An efficient method to build metamodels for sensitivity analysis is developed. It overcomes the need for sampling points in the parametric space, thus, overcoming the curse of dimensionality. Numerical examples in 1D and 2D are presented to show the effectiveness of the method.

Contents

3.1 Introduction

The design process for engineering systems typically involves conducting experiments and extensive simulations. These simulations involve solving complex systems of equations, which can vary in duration from a few minutes to several months, depending on the complexity of the system being analyzed. As the design parameters change, these simulations need to be repeated multiple times to evaluate different design configurations. With an increase in the number of design parameters, the number of simulations required can quickly grow exponentially. This significant computational burden poses a challenge and adds complexity to the design process, potentially impeding its efficiency.

To solve this burden, surrogate or metamodels are used instead of performing full-system simulations. Metamodels are approximate models that are built offline from data generated from experiments or simulations. Statistical methods are used to choose the number and value of the design parameters at which the system will be evaluated. This hugely reduces the computational burden since the number of system evaluations could be small yet approximating the whole parameter space. Evaluating the metamodel for a given design configuration is much cheaper than performing the full-scale simulation, which makes it attractive to be used quickly for the design of systems and process control.

In recent years model order reduction techniques have been developed to build surrogate models taming this curse of dimensionality [1; 2]. These rely on the fundamental assumption that the solution to the parametric problem lies in a low-dimensional manifold of the original subspace where the solution approximation is sought. Learning the structure of this manifold is done through an offline training procedure minimizing the L2 distance from existing data, usually collected from multiple runs of a high-fidelity solver, and produces a low-rank basis that can be reused to represent the solution of new problems for unseen choices of the parameters. In practice, the choice of reduced-basis representation is equivalent to assuming a tensor format for the problem solution. Among the different choices the Canonical Polyadic (CP), Tucker, and Tensor Train (TT) are the most commonly used, as they provide a compact representation of the parametric solution as well as a reduced complexity of the model [3–5].

The presence of a moving flow front in multi-phase flow introduces an additional difficulty to get effective model representations. Tensor formats are generally regarded as unfit to represent solutions exhibiting a moving discontinuity because, due to the dual-scale nature of the problem, there is a need for a large number of basis vectors to obtain a good approximation of the solution. For instance, applying a standard CP space-time decomposition to a simple 1D moving Heaviside function cannot provide an accurate approximation unless a high number of modes is used. In practice, the rank needed to obtain reasonably accurate results is not offering any computational advantage compared to full-order representations. This issue is well documented in the community of model order reduction and it affects not only hyperbolic equations that are likely to give rise to shocks or discontinuities but also

to other problems in which the physics involved produces localized effects in the solutions. The issue was tackled by [6–8] who proposed ways to fix the problem, however, it is still a pressing difficulty that requires attention.

PINN can offer an interesting way to build metamodels since neural networks have a great approximation capability. The main objective of this chapter is to use PINN to build metamodels of the resin injection process introduced in 2.2. We call the classical framework of using PINN to build metamodels META-PINN throughout the chapter. Another strategy is developed to build metamodels for sensitivity analysis problems; we call it SA-PINN (SA stands for sensitivity analysis). The proposed methodology is data-free, meaning that no simulation or experimental data is used in the process of building the model.

This chapter is divided into two main parts. The first one discusses the framework of META-PINN followed by numerical examples. The next part discusses building metamodels for sensitivity analysis problems and introduces SA-PINN. Numerical examples are shown for the second part as well. The chapter ends with a summary and conclusion.

3.2 META-PINN

3.2.1 Framework

To build metamodels in PINN, extra parameters of interest (λ) are added as inputs to the neural network along with the spatial ($\mathbf{X}$) and temporal (t) inputs. The collocation points are then sampled in space-time and parameter of interest space. The rest of the methodology is exactly the same as in training PINN for solving forward problems. The PINN architecture, shown in figure 2.2, is modified so as to include the parameter of interest λ and can be seen in blue color in figure 3.1.

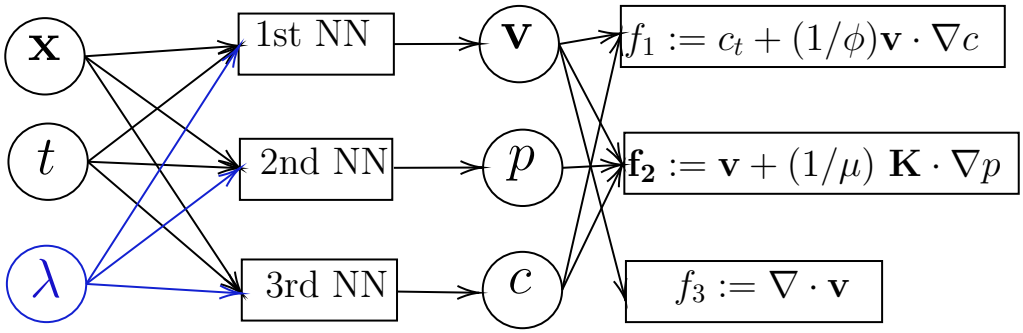

Figure 3.1: META-PINN structure for a general metamodel of a transient two-phase flow in porous media problem.

The loss function can, then, be defined as follows:

$$Loss = \lambda_v\ loss_v + \lambda_c\ loss_c + \lambda_p\ loss_p + \lambda_1\ loss_{f_1} + \lambda_2\ loss_{f_2} + \lambda_3\ loss_{f_3} \quad (3.1)$$

where

$$loss_v = \frac{1}{N_v}\sum_{i=1}^{N_v} r_v^2(\mathbf{x_v}^i, t_v^i, \lambda_v^i) = \frac{1}{N_v}\sum_{i=1}^{N_v} ||\mathbf{v}(\mathbf{x_v}^i, t_v^i, \lambda_v^i)\cdot\mathbf{n}(\mathbf{x_v}^i, t_v^i, \lambda_v^i)||^2, \tag{3.2}$$

$$loss_c = \frac{1}{N_c}\sum_{i=1}^{N_c} r_c^2(\mathbf{x_c}^i, t_c^i, \lambda_c^i) = \frac{1}{N_c}\sum_{i=1}^{N_c} ||c(\mathbf{x_c}^i, t_c^i, \lambda_c^i) - c_b^i||^2, \tag{3.3}$$

$$loss_p = \frac{1}{N_p}\sum_{i=1}^{N_p} r_p^2(\mathbf{x_p}^i, t_p^i, \lambda_p^i) = \frac{1}{N_p}\sum_{i=1}^{N_p} ||p(\mathbf{x_p}^i, t_p^i, \lambda_p^i) - p_b^i||^2, \tag{3.4}$$

$$loss_{f1} = \frac{1}{N_{f1}}\sum_{i=1}^{N_{f1}} ||f_1(\mathbf{x_{f1}}^i, t_{f1}^i, \lambda_{f1}^i)||^2 = \frac{1}{N_{f1}}\sum_{i=1}^{N_{f1}} ||c_t + (1/\phi)\mathbf{v}\cdot\nabla c||^2_{(\mathbf{x_{f1}}^i, t_{f1}^i, \lambda_{f1}^i)}, \tag{3.5}$$

$$loss_{f2} = \frac{1}{N_{f2}}\sum_{i=1}^{N_{f2}} ||f_2(\mathbf{x_{f2}}^i, t_{f2}^i, \lambda_{f2}^i)||^2 = \frac{1}{N_{f2}}\sum_{i=1}^{N_{f2}} ||\mathbf{v} + \frac{1}{\mu}\mathbf{K}\cdot\nabla p||^2_{(\mathbf{x_{f2}}^i, t_{f2}^i, \lambda_{f2}^i)}, \tag{3.6}$$

$$loss_{f3} = \frac{1}{N_{f3}}\sum_{i=1}^{N_{f3}} ||f_3(\mathbf{x_{f3}}^i, t_{f3}^i, \lambda_{f3}^i)||^2 = \frac{1}{N_{f3}}\sum_{i=1}^{N_{f3}} ||\nabla\cdot\mathbf{v}||^2_{(\mathbf{x_{f3}}^i, t_{f3}^i, \lambda_{f3}^i)}, \tag{3.7}$$

and $\{\mathbf{x_v}^i, t_v^i, \lambda_v^i, v_b^i\}_{i=1}^{N_v}$, $\{\mathbf{x_c}^i, t_c^i, \lambda_c^i, c_b^i\}_{i=1}^{N_c}$ and $\mathbf{x_p}^i, t_p^i, \lambda_p^i, p_b^i\}_{i=1}^{N_p}$ are the points where the initial/boundary conditions are defined for $\mathbf{v}$, c and p, respectively. While $\{\mathbf{x_{f1}}^i, t_{f1}^i, \lambda_{f1}^i\}_{i=1}^{N_{f1}}$, $\{\mathbf{x_{f2}}^i, t_{f2}^i, \lambda_{f2}^i\}_{i=1}^{N_{f2}}$ and $\{\mathbf{x_{f3}}^i, t_{f3}^i, \lambda_{f3}^i\}_{i=1}^{N_{f3}}$ are the collocation points in space, time, and parameter of interest domain for the three residuals, respectively, where the physics are enforced, and λ_i are the weights of each term in the loss function.

3.2.2 Numerical examples

One-dimensional injection

We consider the one-dimensional injection problem introduced in section 2.5.1. The same parameters introduced in table 2.1 are used except for the total simulation time, T, which is set to 1 in this example. The permeability, k, is chosen as the parameter of interest in this example. k is chosen to lie between 0.5 and 2 ($k \in [0.5, 2]$).

The proposed adaptivity algorithm (section 2.4) is extended for metamodeling problems and will be used for all the following examples. The results from using the adaptivity algorithm along with PINN will be simply referred to as "PINN" in this chapter.

For training, Adam optimizer is used in the beginning following with BFGS. This optimization strategy will be followed in all the following examples in this chapter. In this example, 1000 Adam iterations were performed followed by around a 1000 BFGS iterations. The loss function contributions vs. the number of iterations is shown in figure 3.2.

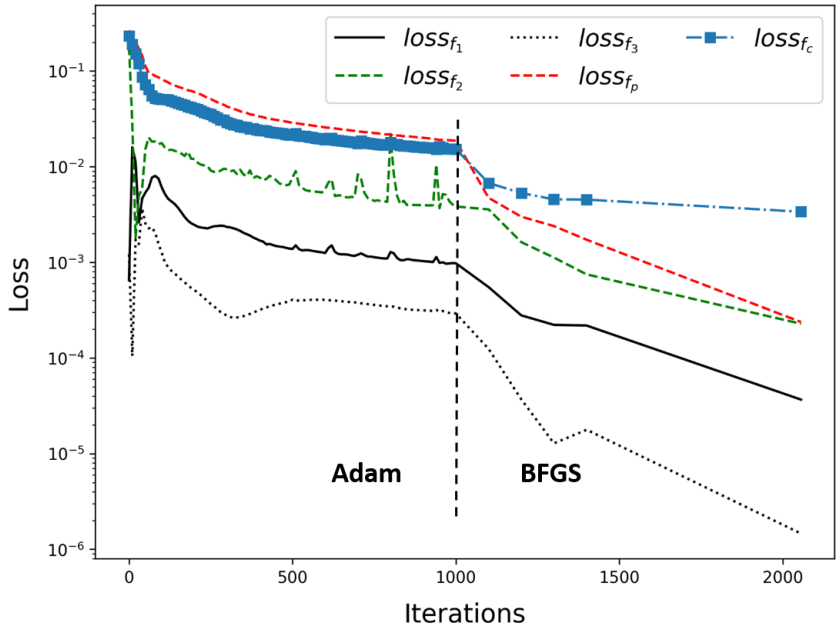

Figure 3.2: Loss function contribution for the 1D filling example vs. the number of iterations.

As can be seen from figure 3.2, all the loss contributions are converging with a higher convergence rate during BFGS phase of training. The kinks in the loss function are due to adaptation of collocation points during the training. The flow front position is, then, extracted from the fraction function field for different values of k which are $0.5, 0.75, 1.0, 1.5$, and 2.0. Figure 3.3 shows the results of the metamodel (PINN) along with the analytical solution.

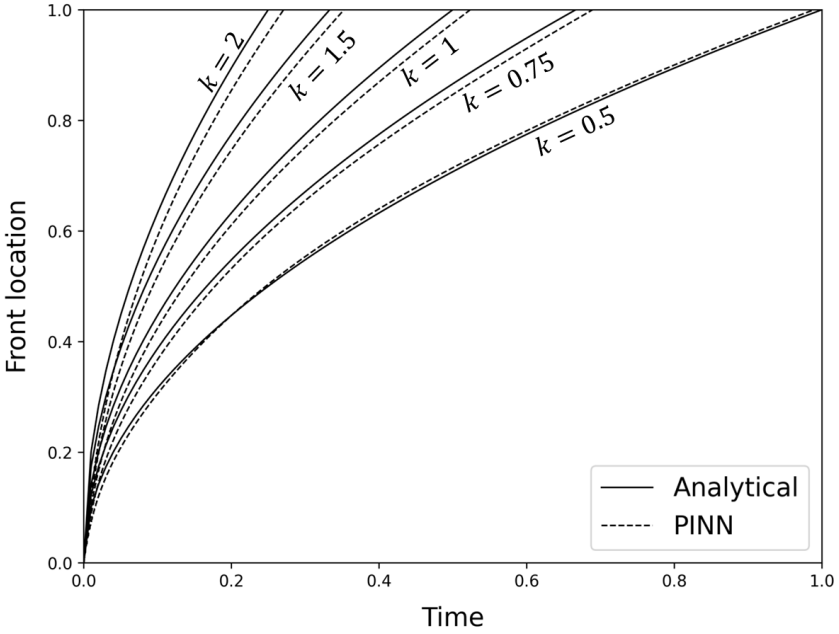

Figure 3.3: Front position with time for $k = 0.5, 0.75, 1, 1.5, 2$ using META-PINN along with the analytical solution.

We can see from figure 3.3 that the built metamodel is able to predict the flow front position but with lower accuracy for some cases such as at $k = 2$. The reason for this reduced accuracy might be due to the reduced number of collocation points used at the start of the training process.

Two-dimensional central injection

The next studied example is the two-dimensional central injection problem introduced in section 2.5.2. The same parameters are used as in section 2.5.2. The principal permeability in the x direction, k_{xx}, is chosen as the parameter of interest in this example. Its values are chosen to lie between 1 and 2 ($k_{xx} \in [1, 2]$).

The fraction function is plotted for 3 scenarios, ($k_{xx} = 1$, $k_{xx} = 1.5$, and $k_{xx} = 2$). The results are shown in figure 3.4.

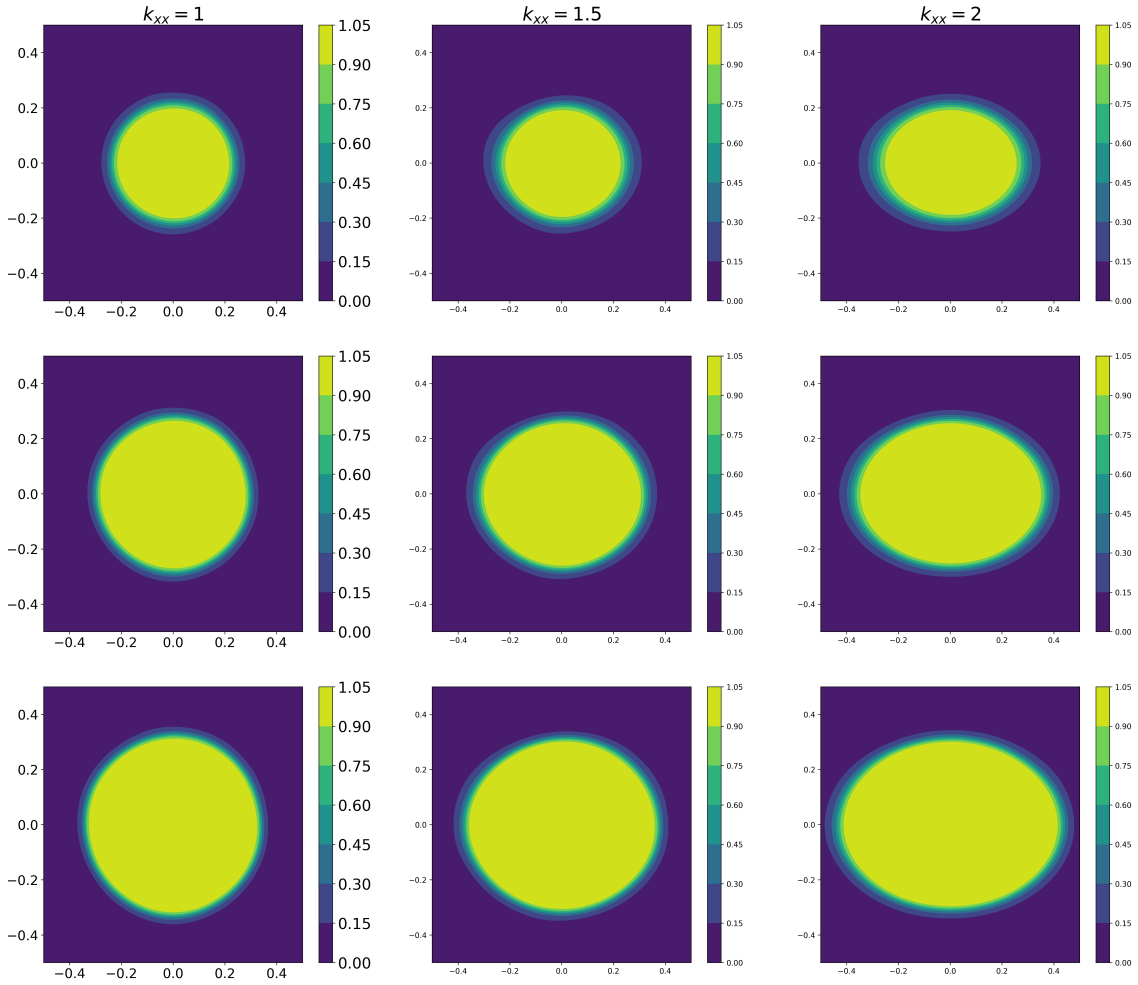

Figure 3.4: Fraction function for the central injection example at three different instants (vertically) for $k_{xx} = 1$, 1.5, and 2, from left to right respectively.

2D injection problem with defect

This example deals with injection in a square mold with side unity. The upper and lower boundaries are impermeable (velocity in the y-direction is zero corresponding

to $\mathbf{v} \cdot \mathbf{n} = 0$). The left edge acts as an inlet with a pressure set to unity and the right edge is an outlet with zero pressure. The full domain has an isotropic permeability k_1 equal to 1 except for a square in the center of side length 0.3 with permeability k_2 ranging from 0.1 to 1. The problem description and domain are shown in figure 3.5.

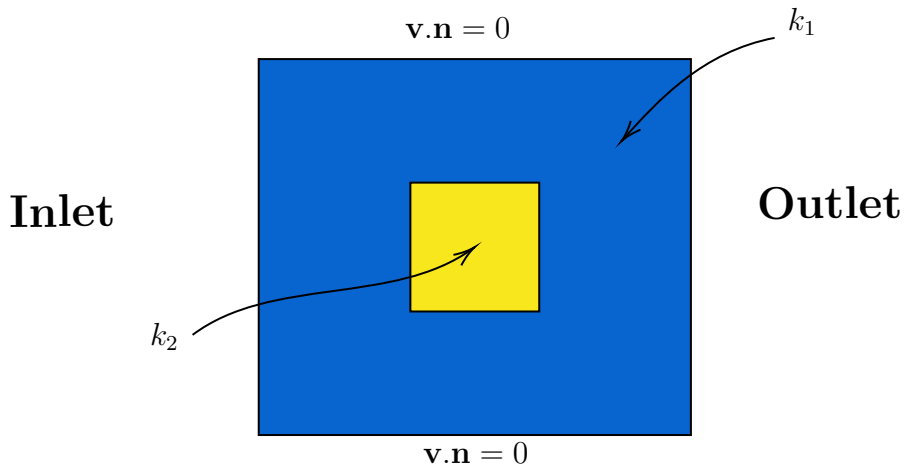

Figure 3.5: Domain and problem description of the 2D injection example with a defect represented by different permeability in a square in the center.

The fraction function is plotted for values of k_2 0.1, 0.5, and 1 in figure 3.6.

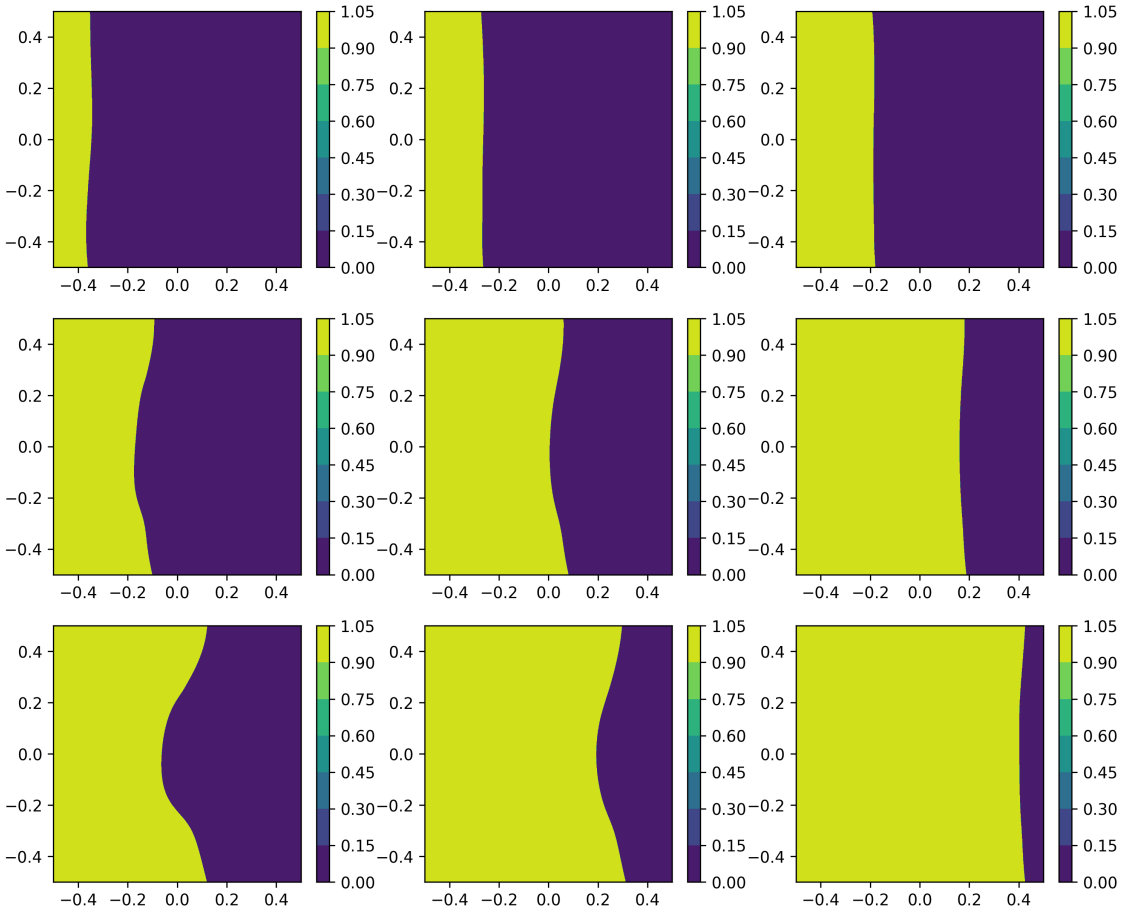

Figure 3.6: Fraction function at three different instants for $k_2 = 0.1$, 0.5, and 1.

Variable inlet gate location

In this example, we study the injection in a square mold with a side length of unity. The problem domain and description are shown in figure 3.7. The upper and lower edges are impermeable (velocity in the y-direction is zero). The right edge is an outlet with pressure set to zero. The left edge includes an inlet of size 0.2 where pressure is fixed to unity. The location of the inlet center y_{in} is chosen to be a parameter of interest, meaning that it can move along the left edge. The rest of the edge is impermeable (velocity in the x-direction is zero). The rest of the problem parameters are chosen as in table 2.1.

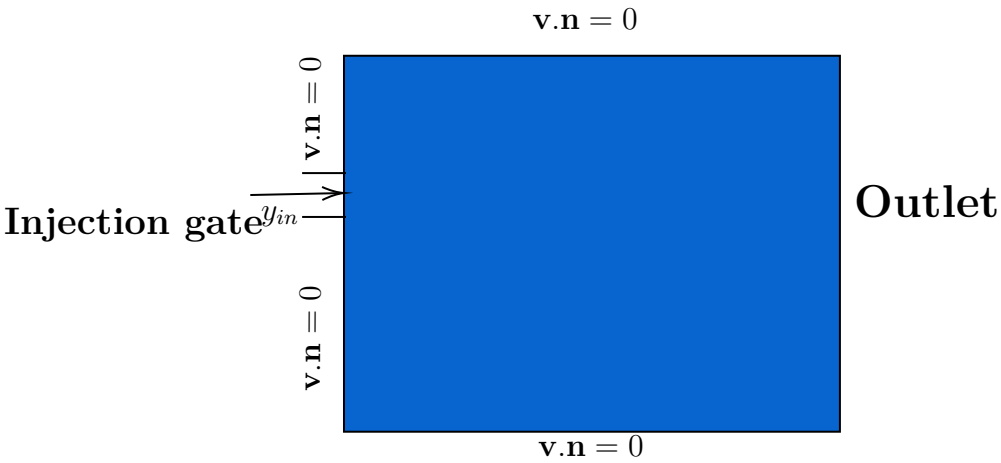

Figure 3.7: Domain and problem description of the 2D injection example with variable inlet location.

In this test, the boundary conditions related to the upper and lower boundaries are treated by hard enforcement. This is done by choosing the following approximation to y component of the velocity:

$$v_y = (y + 0.5)(y - 0.5)v_{NN}, \tag{3.8}$$

where v_{NN} is a neural network. Through this approximation, the upper and lower boundary conditions are enforced by construction and there is no need to add the related terms in the loss function. The rest of the conditions are treated as classical PINN by adding the related terms to the loss function. By this enforcement, the number of terms in the combined loss function is reduced, leading to an easier minimization task for the optimizer.

The fraction function is plotted for y_{in} values of 0, 0.2, and 0.4 at several instants to show the evolution of the flow and the effect of the location of the injection gate. This can be seen in figure 3.8.

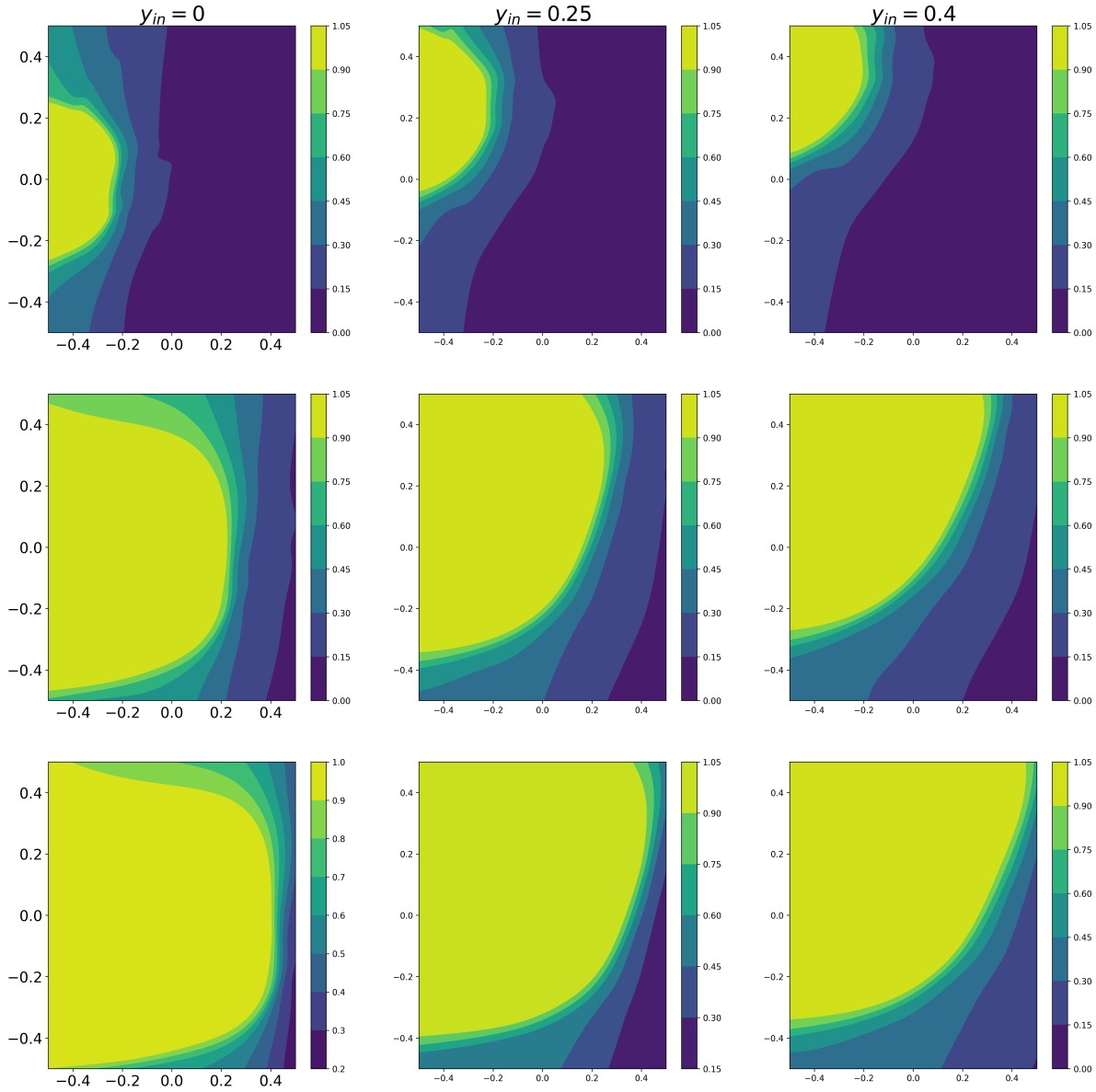

Figure 3.8: Fraction function at three different instants vertically and horizontally for $y_{in} = 0$, 0.25, and 0.4, respectively.

This example can be applied to process design by studying the effect of the inlet gate location on a function of interest such as the filling time. By that, we can choose the best gate location that reduces mold filling time.

We have seen from the previous examples the ability of PINN to build metamodels in resin injection in fibrous media. Even though the results of the fraction function are not accurate as in figure 3.8; this can be improved by adding more collocation points.

This problem offers challenges when being faced with classical model order reduction techniques or the adjoint method [9]. That is mainly due to the existence of a moving discontinuity. However, PINN did not face those challenges in dealing with such discontinuity.

3.3 Sensitivity Analysis with PINN (SA-PINN)

The main objective of PINN is to find a solution that minimizes the residual of the PDE within the spatiotemporal domain that is represented by the collocation points while respecting the initial and boundary conditions. The result is a solution $\hat{u}(t, \mathbf{x}; \hat{\mu})$ to the PDE at a specific value $\hat{\mu}$. To build a metamodel, which is useful for performing sensitivity analysis or optimizing a process, the structure of the neural network is changed to accommodate for another input which is the parameter of interest μ. Then, one adds collocation points in the spatiotemporal-parametric space as described in section 3.2.

The main issue of building such models is that the number of collocation points grows exponentially with the number of parameters of interest. The problem can, then, easily become computationally intractable if there are several parameters of interest, which is common in most engineering applications.

To address this challenge, an alternative approach is proposed. Similarly to building parametric models, the structure of the network is modified to incorporate extra inputs representing the input parameters of interest. The collocation points are kept in the spatio-temporal domain and no points are added in the parametric space. Instead of solely minimizing the loss function, representing the residual of the PDE and the conditions, the derivative of the loss function with respect to the parameter of interest is also minimized. The modified loss function will then be formulated as the sum of the residual, the derivative of the residual with respect to μ, and terms related to satisfying the initial and boundary conditions. By employing this technique, the solution can be accurately determined within a small neighborhood of $\hat{\mu}$.

3.3.1 Formulation

The technique can be summarized in steps as follows:

1. Choose the neural network to have inputs related to space, time, and parameters of interest.
2. Sample the collocation points only in space and time, no need to add points in the parametric space.
3. Create the loss function having terms related to PDE residual, the residual derivative with respect to the parameter of interest, and the terms related to the initial and boundary conditions.

The general example introduced in section 1.4 is taken into consideration to explain the technique. The loss function to solve the problem will be modified for SA-PINN to become

$$Loss = Loss_{PINN} + Loss_{SA-PINN}, \tag{3.9}$$

where

$$Loss_{PINN} = \lambda_0\ loss_0 + \lambda_D\ loss_D + \lambda_N\ loss_N + \lambda_r\ loss_r, \tag{3.10}$$

λ_i the weights for each loss term which play an important role in the optimization process,

$$loss_0 = \frac{1}{N_0}\sum_{i=1}^{N_0} r_0^2(t_0^i, \mathbf{x}_0^i, \hat{\mu}) = \frac{1}{N_0}\sum_{i=1}^{N_0} ||u(t_0^i, \mathbf{x}_0^i, \hat{\mu}) - u_0^i||^2, \tag{3.11}$$

$$loss_D = \frac{1}{N_D}\sum_{i=1}^{N_D} r_D^2(t_D^i, \mathbf{x}_D^i, \hat{\mu}) = \frac{1}{N_D}\sum_{i=1}^{N_D} ||u(t_D^i, \mathbf{x}_D^i, \hat{\mu}) - u_D^i||^2, \tag{3.12}$$

$$loss_N = \frac{1}{N_N}\sum_{i=1}^{N_N} r_N^2(t_N^i, \mathbf{x}_N^i, \hat{\mu}) = \frac{1}{N_N}\sum_{i=1}^{N_N} ||B(u(t_N^i, \mathbf{x}_N^i, \hat{\mu})) - f_N^i||^2, \tag{3.13}$$

$$loss_r = \frac{1}{N_r}\sum_{i=1}^{N_r} ||r(t_r^i, \mathbf{x}_r^i, \hat{\mu})||^2 = \frac{1}{N_r}\sum_{i=1}^{N_r} ||u_t + \hat{\mu}L(u)||^2_{(t_r^i, \mathbf{x}_r^i)}. \tag{3.14}$$

The indices stand for initial conditions (0), Dirichlet boundary conditions (D), Neumann Boundary conditions (N), and the PDE residual (r).

While

$$Loss_{SA-PINN} = \lambda_{0\mu}\ loss_{0\mu} + \lambda_{D\mu}\ loss_{D\mu} + \lambda_{N\mu}\ loss_{N\mu} + \lambda_{r\mu}\ loss_{r\mu}, \tag{3.15}$$

where

$$loss_{0\mu} = \frac{1}{N_0}\sum_{i=1}^{N_0} \frac{\partial r_0(t_0^i, \mathbf{x}_0^i, \hat{\mu})}{\partial \mu}, \tag{3.16}$$

$$loss_{D\mu} = \frac{1}{N_D}\sum_{i=1}^{N_D} \frac{\partial r_D(t_D^i, \mathbf{x}_D^i, \hat{\mu})}{\partial \mu}, \tag{3.17}$$

$$loss_{N\mu} = \frac{1}{N_N}\sum_{i=1}^{N_N} \frac{\partial r_N(t_N^i, \mathbf{x}_N^i, \hat{\mu})}{\partial \mu}, \tag{3.18}$$

$$loss_{r\mu} = \frac{1}{N_r}\sum_{i=1}^{N_r} \frac{\partial r(t_r^i, \mathbf{x}_r^i, \hat{\mu})}{\partial \mu}. \tag{3.19}$$

Figure 3.9 shows a diagram that summarizes the methodology of SA-PINN. The parts in orange are the added parts from classical PINN. The $u - \hat{u}$ term represents the mismatch of the solution from the initial and boundary conditions. It must be noted that we sample the collocation points only in space and time, but the points have another coordinate μ and all have a nominal value $\hat{\mu}$.

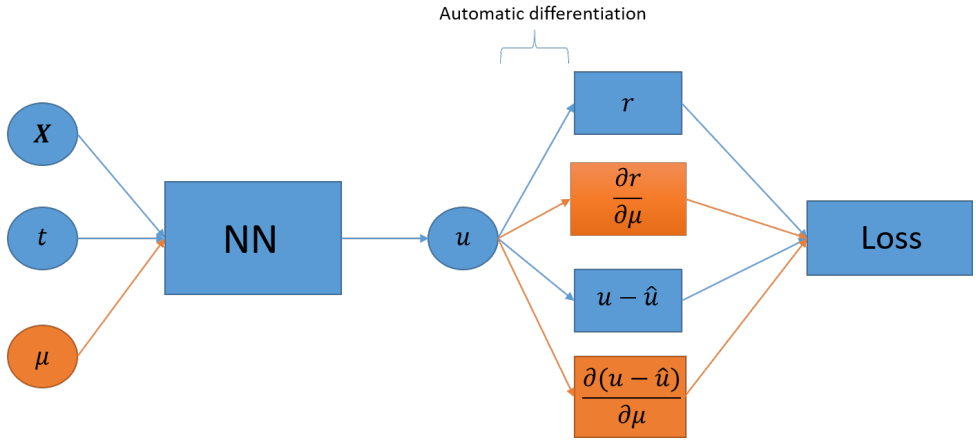

Figure 3.9: Diagram explaining the methodology of SA-PINN.

3.3.2 Numerical examples

3.3.2.1 1D diffusion-advection equation

The first example is a steady one-dimensional diffusion-advection equation where we would like to study the effect of perturbations in the diffusion term ϵ on the solution. The strong form of the problem can be written as follows:

$$\begin{aligned} \epsilon\, u_{xx} - u_x + 1 = 0, \quad x \in [0,1], \\ u(0) = 1, \quad u(1) = 3 \end{aligned} \tag{3.20}$$

The chosen nominal value for ϵ is 0.1. u_{xx} and u_x are respectively the second and first-order derivatives of the solution u. The loss function is written as:

$$Loss = \lambda_f\ loss_f + \lambda_b\ loss_b + \lambda_{f_\epsilon}\ loss_{f_\epsilon} + \lambda_{b_\epsilon}\ loss_{b_\epsilon}, \tag{3.21}$$

where

$$loss_f = \frac{1}{N_f}\sum_{i=1}^{N_f} ||r(x_f^i, \hat{\epsilon})||^2 = \frac{1}{N_f}\sum_{i=1}^{N_f} ||\hat{\epsilon}\ u_{xx} - u_x + 1||^2_{x_f^i}, \tag{3.22}$$

$$loss_b = \frac{1}{N_b}\sum_{i=1}^{N_b} r_b^2(x_b^i, \hat{\epsilon}) = \frac{1}{N_b}\sum_{i=1}^{N_b} ||u(x_b^i, \hat{\epsilon}) - u_b^i||^2, \tag{3.23}$$

$$loss_{f_\epsilon} = \frac{1}{N_f}\sum_{i=1}^{N_f} \frac{\partial r}{\partial \epsilon}^2_{(x_f^i, \hat{\epsilon})}, \tag{3.24}$$

$$loss_{b_\epsilon} = \frac{1}{N_b}\sum_{i=1}^{N_b} \frac{\partial r_b}{\partial \epsilon}^2_{(x_b^i, \hat{\epsilon})}. \tag{3.25}$$

The weights for the different terms in the loss function are set to 1 for the original PINN terms (λ_f and λ_b) and 0.1 for the added sensitivity terms (λ_{f_ϵ} and λ_{b_ϵ}).

For training, 1000 Adam iterations are performed followed by 250 BFGS iterations. The loss function contributions vs. the number of iterations is shown in figure 3.10. In the figure, $loss_f$ represents the loss due to the PDE residual, $loss_b$ the loss due to the boundary conditions, while $loss_{f_\epsilon}$ and $loss_{b_\epsilon}$ represents the added terms for SA-PINN.

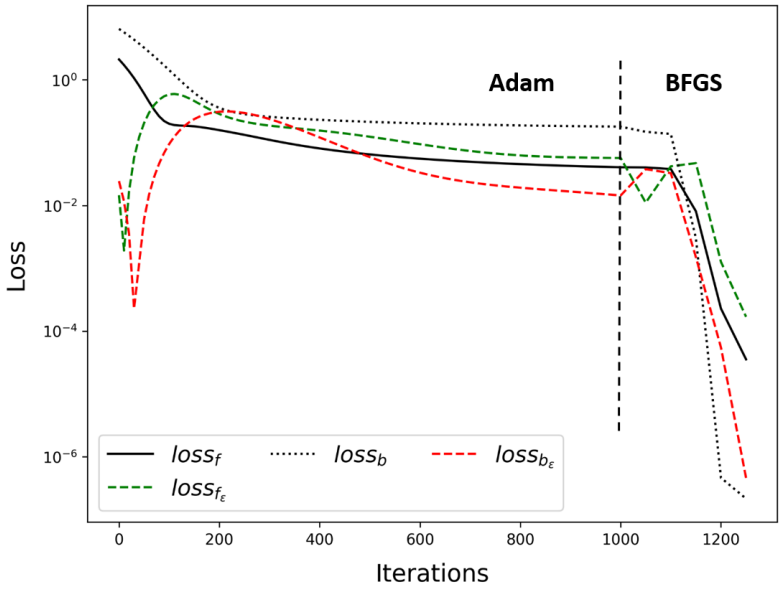

Figure 3.10: Loss function contribution for the 1D diffusion-advection example vs. the number of iterations.

The solution u using PINN and SA-PINN is shown in figure 3.11 along with the analytical solution for $\epsilon = 0.1$.

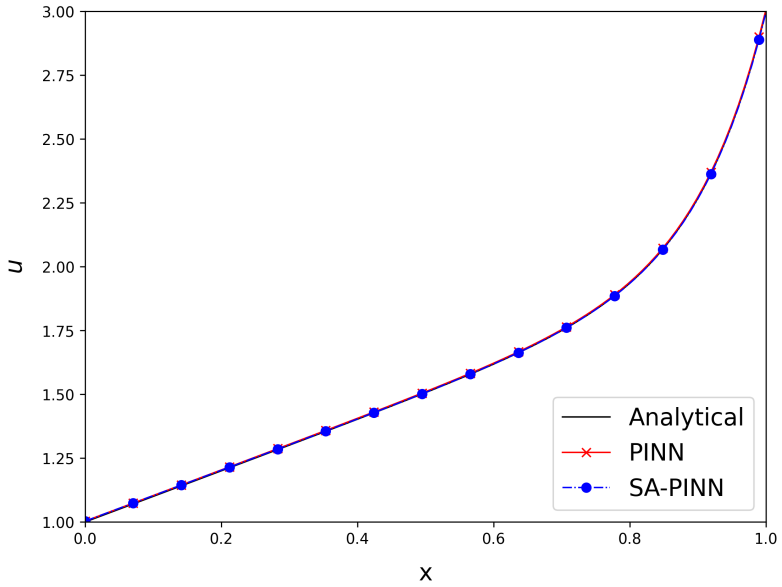

Figure 3.11: Solution u at $\epsilon = 0.1$ using PINN and SA-PINN along with the analytical solution of the 1D advection-diffusion problem.

From figure 3.11, we can see that PINN and SA-PINN accurately capture the analytical solution of the problem. The derivative of the solution with respect to ϵ, $\partial u/\partial \epsilon$, at $\epsilon = 0.1$ is shown in figure 3.12.

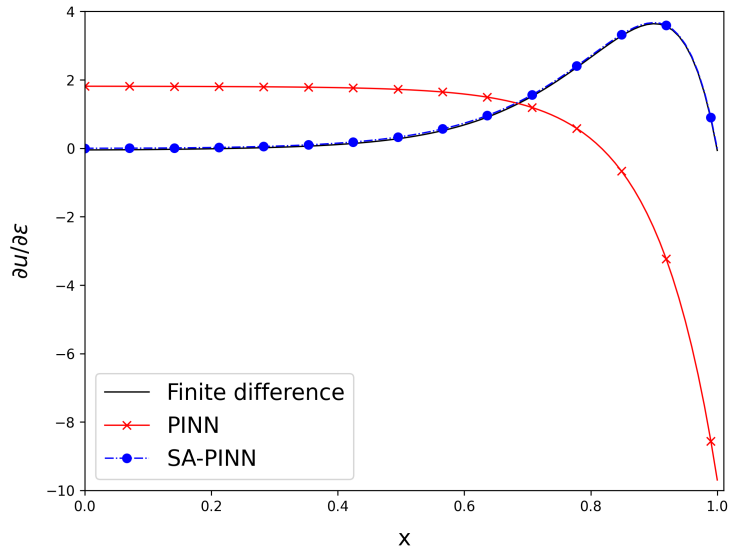

Figure 3.12: $\frac{\partial u}{\partial \epsilon}$ at $\epsilon = 0.1$ using PINN and SA-PINN along with the finite difference solution of the 1D advection-diffusion problem.

The reference finite difference solution in figure 3.12 is obtained by obtaining three different PINN solutions near $\epsilon = 0.1$ and then calculating the derivative using central finite difference. We can see that classical PINN fails to predict the derivative, while, SA-PINN accurately predicts the derivative due to the added regularization term in the loss function. The loss function for different values of ϵ is plotted in figure 3.13 for PINN and SA-PINN.

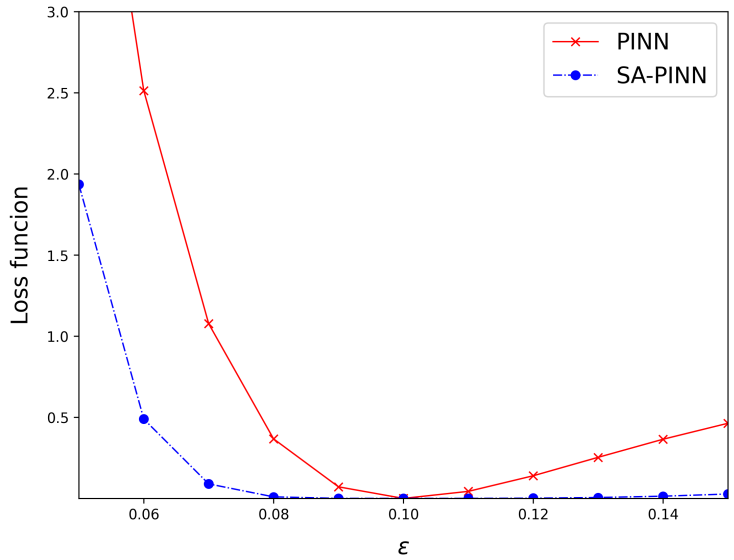

Figure 3.13: Loss function for different ϵ values using PINN and SA-PINN of the 1D advection-diffusion problem.

As seen in figure 3.13, SA-PINN has the effect of greatly flattening the loss curve in a neighborhood near the nominal value of $\epsilon = 0.1$. This leads to better solutions than PINN in the neighborhood and accurate derivative calculation at $\epsilon = 0.1$.

3.3.2.2 2D Poisson's problem

The next example is a 2-dimensional Poisson's problem where we have multiple parameters to study their effect on the solution. The domain Ω is shown in figure 3.14 where there exists 9 subdomains each having different diffusivity value.

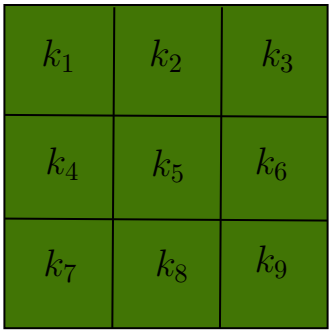

Figure 3.14: 2D Poisson's problem domain Ω with 9 subdomains of different diffusivity values k.

The strong form of the problem can be written as:

$$\begin{aligned} k\ \Delta u &= -1, \quad in\ \Omega, \\ u &= 0, \quad on\ \partial\Omega \end{aligned} \tag{3.26}$$

where Ω is a square with unit sides and k is the diffusivity. The 9 subdomains have equal areas. The diffusivity parameters have the same nominal value which is 1; $\hat{k_1} = \hat{k_2} = ... = \hat{k_9} = 1$.

The approximation space is chosen such that the boundary conditions are satisfied automatically. The approximation reads as follows:

$$\hat{u} = xy(x-1)(y-1)\ u_{NN}(x, y, k_1, ..., k_9) \tag{3.27}$$

where u_{NN} is a neural network with inputs x and y and the diffusivity parameters. The full loss function will then reads as:

$$Loss = \lambda_f\ loss_f + \sum_{i=1}^{9} \lambda_i\ loss_{f_{k_i}}, \tag{3.28}$$

where $\lambda_f = 1$, all values of λ_i are set to 0.1,

$$loss_f = \frac{1}{N_f} \sum_{i=1}^{N_f} ||r(x_f^i, \hat{k})||^2 = \frac{1}{N_f} \sum_{i=1}^{N_f} ||\hat{k}\ \Delta u + 1||_{x_f^i}^2, \tag{3.29}$$

and

$$loss_{f_{k_i}} = \frac{1}{N_f} \sum_{i=1}^{N_f} \frac{\partial r}{\partial k_i}^2 {}_{(x_f^i, \hat{k_i})}. \tag{3.30}$$

The different loss terms are plotted against the iterations in figure 3.15.

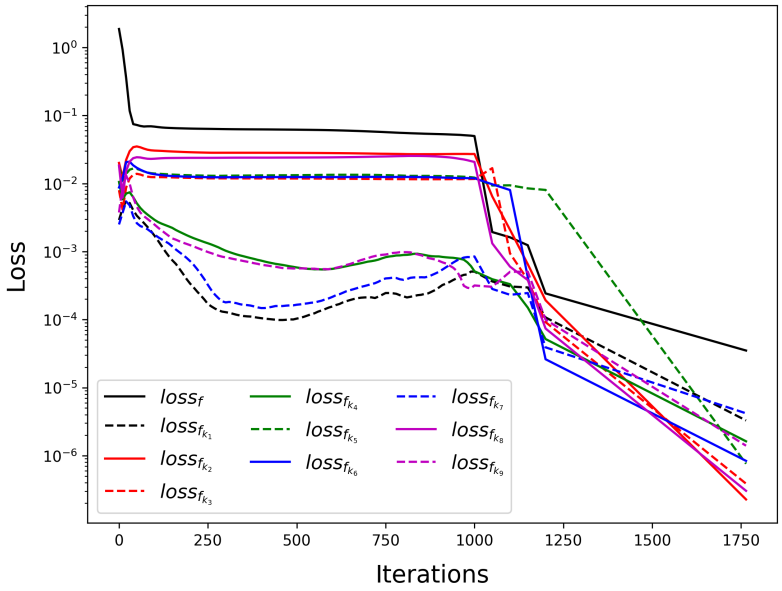

Figure 3.15: Loss function contribution for the 2D Poisson's example vs. the number of iterations.

The PINN solution of the boundary value problem is shown in figure 3.16. The solution appears to be accurate and agrees with the analytical solution of the problem.

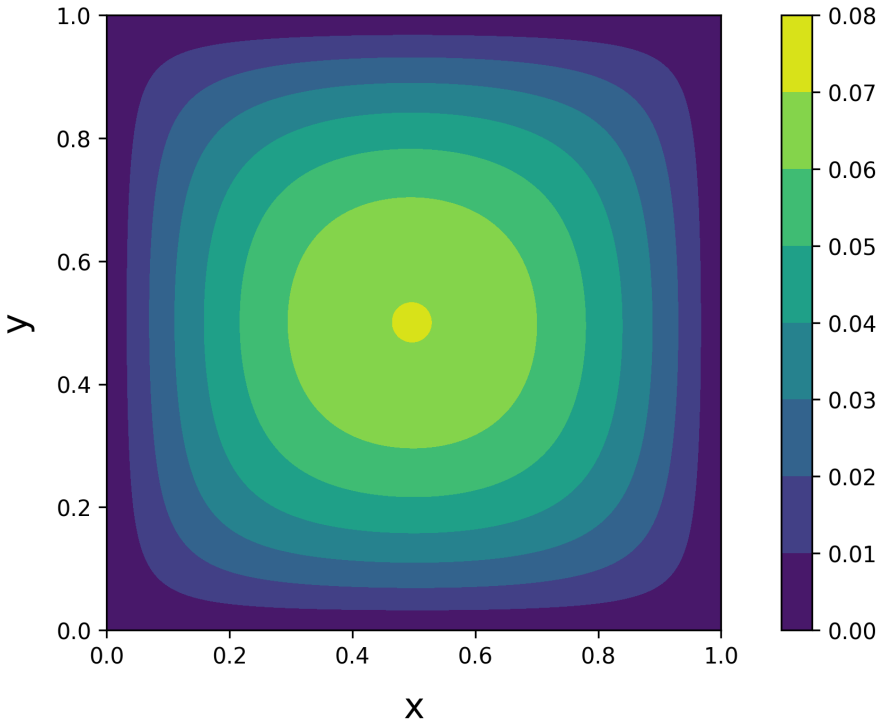

Figure 3.16: PINN solution of the 2D Poisson's boundary value problem.

The sensitivity terms $\frac{\partial u}{\partial k_i}$ can then be plotted to see the effect of the diffusivity on the solution.

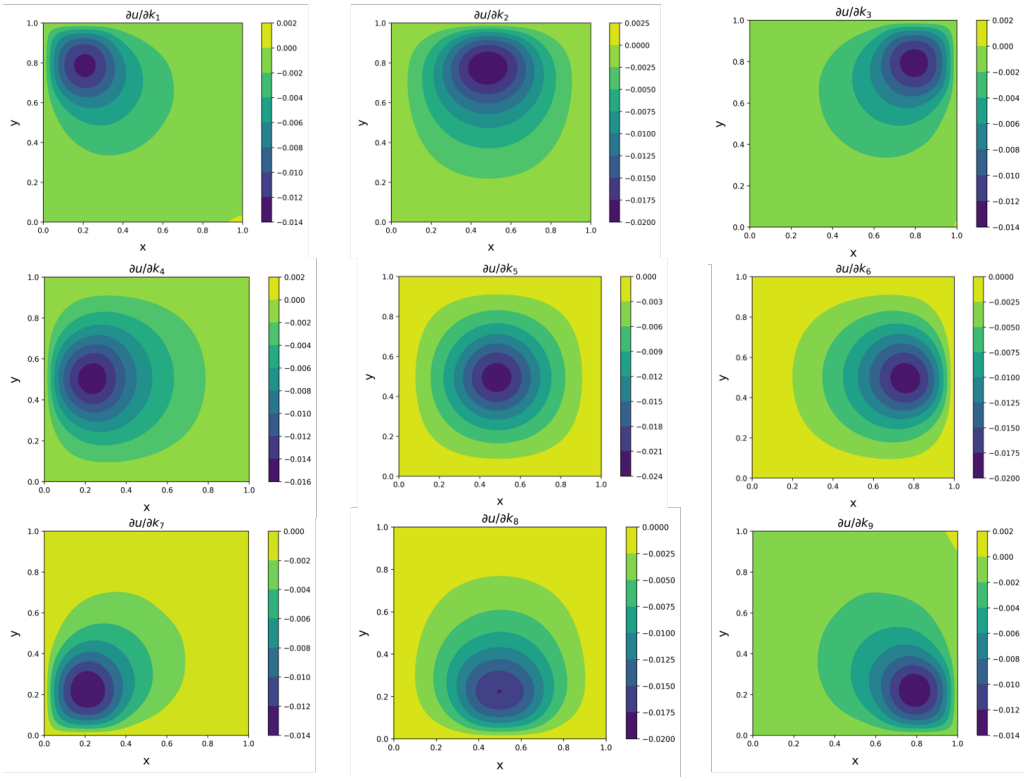

Figure 3.17: Different derivatives $\frac{\partial u}{\partial k_i}$ of the solution with respect to k_i of the 2D Poisson's problem.

The computational time is plotted versus the number of parameters with respect to which sensitivity terms are added in figure 3.18.

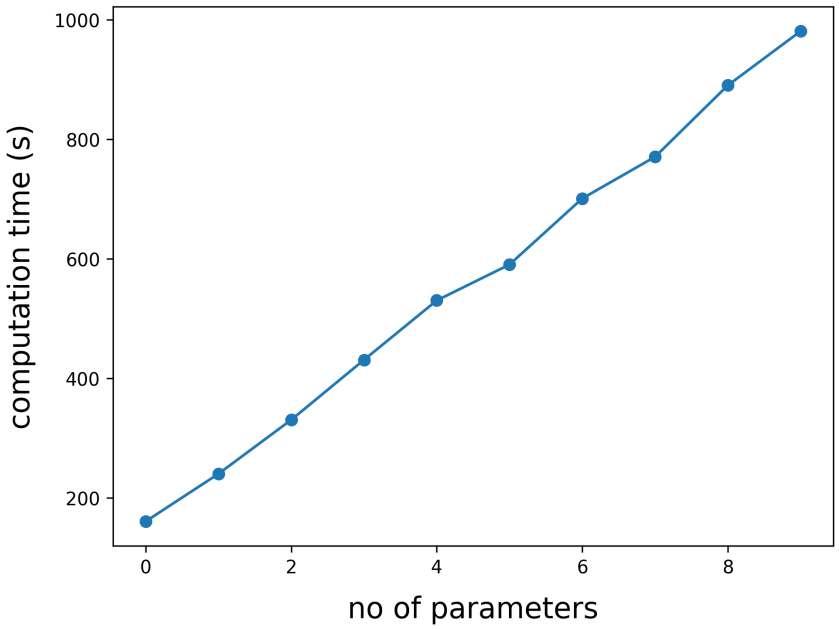

Figure 3.18: Computational time vs. number of sensitivity parameters.

It can be seen from figure 3.18 that the computational time grows linearly when increasing the number of parameters the sensitivity is calculated with respect to. This happens because the number of collocation points is the same when adding a new term to the loss function; as a consequence, the added cost is the same when adding new sensitivity terms. This is a great advantage of the proposed approach because if someone adds a parameter to study the sensitivity with META-PINN, the collocation points number can grow exponentially and so as the computational time.

3.3.2.3 1D transient two-phase flow in porous media

We consider the one-dimensional injection problem introduced in section 2.5.1. The parameter of interest in this case is the permeability, k. The chosen nominal value of the study is $\hat{k} = 1$.

The loss function terms are plotted against the iterations in figure 3.19. In the figure, l_1, l_2, l_3, l_p, and l_c refer to the loss functions referring to the advection of fraction function, Darcy's equation, mass conservation residuals, pressure, and fraction function boundary conditions. The rest of the terms with the added g refer to the corresponding terms added by SA-PINN.

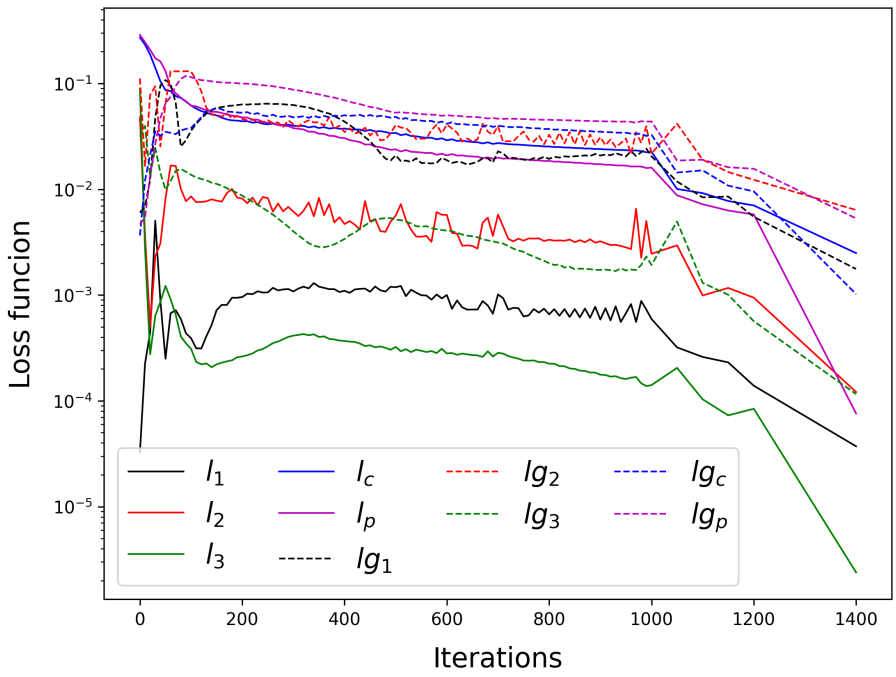

Figure 3.19: Loss function contributions vs. the number of iterations for the 1D two-phase flow in porous media example.

First, we plot the front location for three different values of k ($k = 1$, 0.50, and 2) by taking the 0.5 level set of the fraction function c. This is plotted in figure 3.20. In each figure, SA-PINN results are plotted against classical PINN along with the analytical solution for comparison.

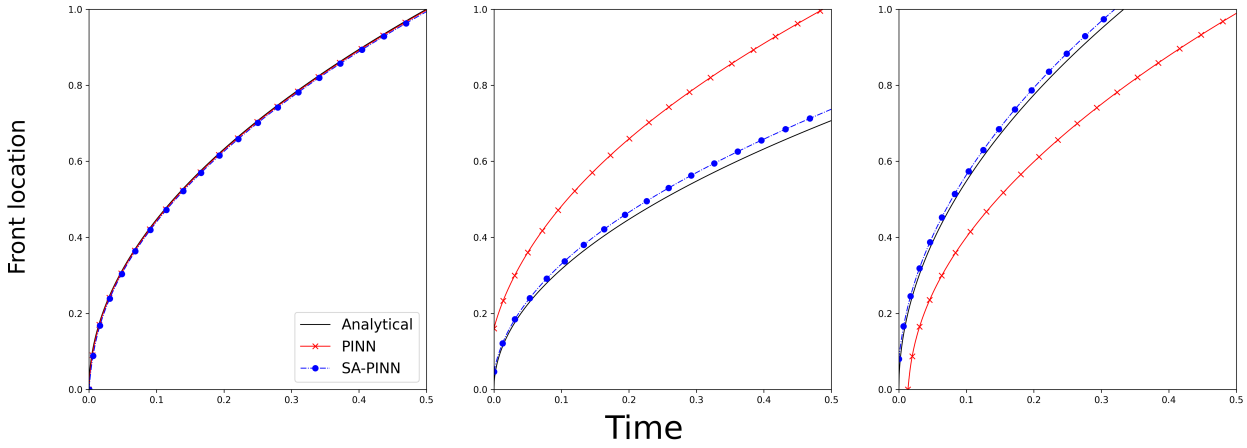

Figure 3.20: Flow front location vs. time for three different values of k ($k = 1$, 0.5 and 2, respectively on the left, center, and right) of the transient two-phase 1D flow in porous media problem.

We can notice that SA-PINN provides good results for values of k away from the nominal value $k = 1$. Classical PINN accurately predicts the solution only at the nominal value ($k = 1$), however, away from that value, inaccurate solutions were obtained which is clear from the two red lines.

In figure 3.21, we plot the time the flow front reaches the position $x = 0.5$ vs. k. We compare the solution from SA-PINN with the analytical solution.

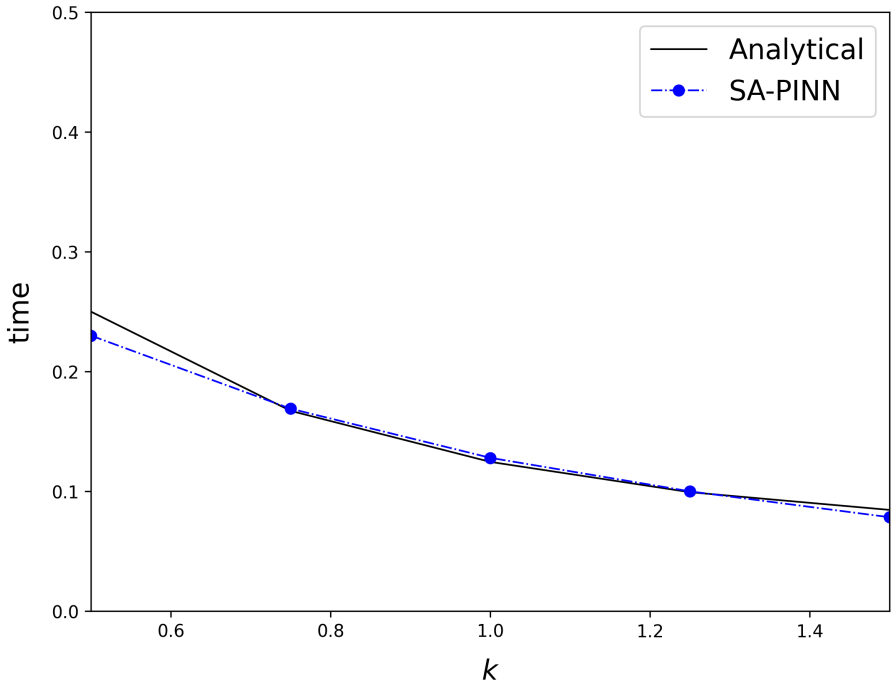

Figure 3.21: Time at which the flow front reaches $x = 0.5$ vs. k for the transient two-phase flow in porous media problem.

We can see a good estimation of the filling time at different values of k using

SA-PINN. This result can be useful in applications of injection processes to estimate the filling time as a function of a parameter of interest.

3.3.3 Discussion

From the examples shown in this chapter, we can see the ability of PINN to build metamodels for problems involving discontinuities (a moving flow front in our case). This is a complicated problem if addressed using widely used methods to build metamodels such as model order reduction techniques. Metamodels can in turn be further used to address online control problems to identify possible process defects or to quickly solve inverse problems.

The issue of computational cost was tackled by the newly developed technique SA-PINN. Through the combined minimization of the residual and its derivative with respect to parameters of interest, we ensure the validity of the solution within a neighborhood of the nominal values of the parameters which allow accurate sensitivity estimation. In figure 3.22, we show the effect of the method by plotting the contours of the term $loss_f$ in equation 3.28 for different k_i values. The plot is done for only 2 parameters for illustration purposes.

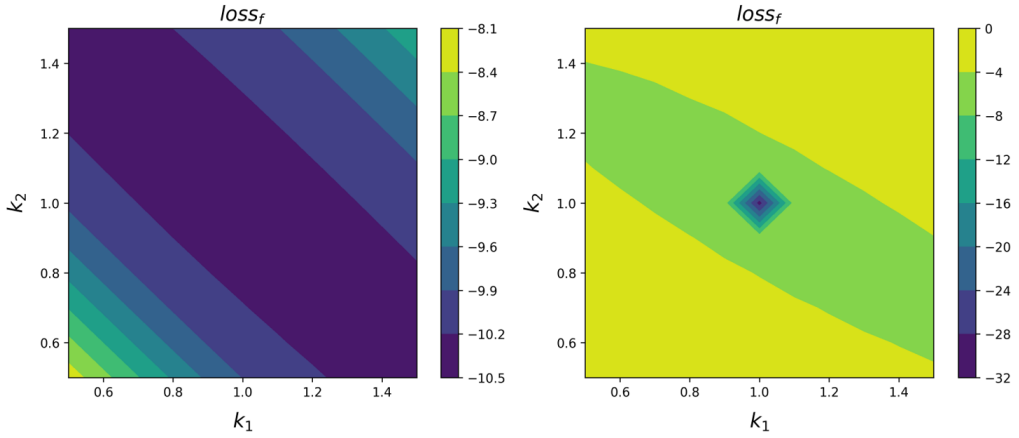

Figure 3.22: The log contours of $loss_f$ for different k_1 and k_2 values ranging from 0.5 to 2 are plotted for the 2D Poisson's example. On the left, the SA-PINN and on the right the PINN solution.

We can see from figure 3.22 that the SA-PINN technique has the effect of greatly reducing the main loss function contribution in a neighborhood near the nominal values of the parameters of interest, thus obtaining good solutions in this neighborhood. On the contrary, PINN has an accurate solution only at the nominal values of the parameters. We showed also that the technique does not face problems in dealing with discontinuities as shown in section 3.3.2.3 which could be useful for several applications involving discontinuities.

3.4 Summary and conclusion

In this chapter, we have seen:

- the need to build metamodels for process optimization and control,
- a framework to build metamodels using PINN for mold filling in fibrous media,
- a new technique to build metamodels to perform sensitivity analysis.

To conclude, PINN has the ability to build metamodels in injection molding in fibrous media. It must be noted that building such models is quite a challenge using model order reduction techniques due to the existence of discontinuities in the solution; however, PINN does not suffer these issues.

The newly developed technique, SA-PINN, to perform sensitivity analysis is efficient in building metamodels in which the computational time grows linearly with the number of parameters of interest. This is done by adding regularization terms representing the derivatives of the loss contribution with respect to the parameters of interest. The main originalities in this chapter are:

- building metamodels for injection in fibrous media problems using PINN.
- developing an efficient technique to perform sensitivity analysis using PINN.

Transition

Building metamodels for homogeneous material parameters or boundary conditions is possible using PINN framework. However, some of the material parameters are not homogenous and it is harder to build metamodels for. This inhomogeneity can create issues in the process if it is not well characterized. For example, in LCM processes, this parameter is the permeability where local features can lead to defects in the process. Characterizing the inhomogeneous permeability field is the main goal of the next chapter.

3.5 Bibliography

[1] Stein Krogstad. A sparse basis pod for model reduction of multiphase compressible flow. In *SPE Reservoir Simulation Symposium*. OnePetro, 2011.

[2] Harshit Bansal, S Rave, Laura Iapichino, Wil HA Schilders, and Nathan van de Wouw. Model order reduction framework for problems with moving discontinuities. In *ENUMATH*, pages 83–91, 2019.

[3] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[4] Ivan V Oseledets, DV Savostianov, and Eugene E Tyrtyshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. *SIAM Journal on Matrix Analysis and Applications*, 30(3):939–956, 2008.

[5] José V Aguado, Domenico Borzacchiello, Kiran S Kollepara, Francisco Chinesta, and Antonio Huerta. Tensor representation of non-linear models using cross approximations. *Journal of scientific computing*, 81(1):22–47, 2019.

[6] David Neron and Pierre Ladeveze. Idelsohns benchmark. Technical report, Technical report, 2013.

[7] Liqian Peng and Kamran Mohseni. Nonlinear model reduction via a locally weighted pod method. *International Journal for Numerical Methods in Engineetming*, 106(5):372–396, 2016.

[8] Andrea Ferrero, Angelo Iollo, and Francesco Larocca. Global and local pod models for the prediction of compressible flows with dg methods. *International Journal for Numerical Methods in Engineering*, 116(5):332–357, 2018.

[9] Niklas Kühl, Jörn Kröger, Martin Siebenborn, Michael Hinze, and Thomas Rung. Adjoint complement to the volume-of-fluid method for immiscible flows. *Journal of Computational Physics*, 440:110411, 2021.

Chapter 4

Permeability field identification from textile images

Abstract

In this chapter, we provide a self-supervised learning framework to approximate the permeability field of 2D fibrous media from textile images. The final aim is to make this approximation process online, in real time. The framework is based on merging physics-informed and convolutional neural networks. Data from 34 central injection experiments are used including flow front images and data from pressure sensor located at the inlet. The provided framework is geometry agnostic since the input to the convolutional network is crops of the textile image which have the same characteristics as crops from any 2D geometry image. To validate the model, permeability field predictions are performed for left-out experiments, then the permeability is used to solve a forward problem (central injection) and compared to experimental results. The simulation results using the predicted permeability field show great match with the experiments as opposed to using an average value of the permeability.

Contents

4.1 Introduction

In fibrous media, local variations in the media can take place during the preform manufacturing and handling. These variations lead to variations in permeability. The variations take place within the same part and also from one part to another. To be able to perform accurate realistic mold injection simulations tailored to each part, the permeability tensor has to be treated as a field varying in space, and this field should be different from one part to another according to the local variations within the part.

However, many existing techniques for determining permeability provide only average approximations over the entire domain, disregarding local effects. Alternatively, some methods focus on identifying permeability at the microstructure level, which is impractical for macro-scale porous media due to computational limitations. Therefore, there is a need for approaches that can capture the local variations in permeability within the porous media while still being applicable at a macroscopic scale.

Empirical models exist to estimate the permeability based on the geometry of the reinforcement. One of the most famous models is the Kozeny-Carmen model [1; 2]. The permeability tensor is estimated as follows:

$$\mathbf{K} = \frac{R^2}{4k}\frac{(1-V_f)^3}{V_f^2}\mathbf{I}, \tag{4.1}$$

where V_f is the fibre-volume fraction, R is the fiber radius, k is called Kozeny constant and $\mathbf{I}$ is the identity tensor. The predicted permeability is isotropic which is obviously false for most fibrous structures. A modification was proposed by [3] where different k constants are chosen for different directions adding the anisotropy into the model.

The previously mentioned models only have the fiber radius as a geometry parameter and the rest of the geometrical parameters are lumped into the constants in the model which need to be identified using experiments. This issue was dealt with by [4] where they developed a more inclusive model for unidirectional reinforcement. The issue still prevails for other types of fibrous media since the geometry of the media is quite complex and it is hard to build models taking into account such geometry.

The empirical laws suffer from not being tailored to the specificity of each reinforcement and do not offer an estimation of the permeability as a field. Experimental methods can are commonly employed to identify the permeability for each specific reinforcement [5]. These methods typically involve controlling either the pressure or velocity and measuring the corresponding parameter. Average values are then calculated, and the permeability is evaluated using Darcy's law. In the context of resin transfer molding, two main categories of experiments are typically conducted: unidirectional flow experiments [6; 7] and radial (central injection) experiments [8; 9]. However, these experimental techniques have limitations, including being time-consuming and challenging to perform. Furthermore, the resulting permeability values obtained from these experiments represent average values over the entire

domain, disregarding local effects within the fibrous media as in the case of empirical laws.

Numerical methods offer an interesting alternative to experimental techniques, as they are typically less complicated and time-consuming. These methods aim to replicate the experimental setup by fixing either the pressure or velocity and measuring the other parameter. Averaging is performed, followed by the application of Darcy's law to determine the permeability. Numerical solvers, based on the continuum approach, are commonly used and involve solving either the Navier-Stokes equations [4; 10] or simplified versions such as the Stokes equations [11; 12]. However, these techniques are more applicable to identifying the permeability of microstructures and become computationally complex when dealing with macroscale fibrous media. Additionally, obtaining knowledge of the porous media domain is crucial, which can be acquired through imaging techniques like tomography. Alternatively, a virtual reconstruction of the porous media can be employed to establish the relationship between the porous media properties and permeability.

Mesoscopic numerical methods, such as the Lattice Boltzmann Method (LBM), are also widely used for determining the permeability of microstructures [13; 14]. These methods are better suited for simulating fluid flow in complex geometries. LBM exhibits significant improvements in computational efficiency compared to conventional Computational Fluid Dynamics (CFD) methods, making it suitable for conducting three-dimensional simulations. Utilizing LBM can be advantageous in extracting the relationship between porous media parameters and permeability by conducting multiple cost-effective numerical experiments and studying the effects of these parameters on permeability. However, it is important to note that although LBM has been extensively used in various studies to establish such relationships, the method is primarily effective in identifying permeability on a microscale level and is impractical for macroscale fibrous media.

Some novel research work has a similar framework to the one adopted in our method. In [15], the authors show the capability of feedforward neural networks and convolutional neural networks (CNN) to directly calculate permeability where the inputs to the feedforward neural network are geometric properties of the porous media and the input to the CNN is porous media images. The authors of [16] offer a framework to predict porosity, permeability, and tortuosity of microstructure porous media from images using a CNN. They built 100,000 virtual porous media generated by random deposition. LBM simulations are then performed on all the images to provide an estimation of the porosity, permeability, and tortuosity. A CNN is finally used for the supervised learning task. In this sense the CNN learns to map images to the simulation results in a way that is computationally inexpensive and much faster than the original model used to generate the dataset.

In [17], a similar framework is presented but is applied to predict the diffusivity of porous media. They generate a big dataset of virtual porous media images, then perform LBM simulations to calculate the diffusivity. Finally, they trained a CNN using the generated dataset.

In [18], the authors developed a physics-informed CNN to predict the permeability from microstructure porous media images. The framework is similar to that of [16] where porous media images are generated using the Voronoi tessellation algorithm. This is followed by LBM simulations of the porous media images. Finally, a physics-informed CNN is used for the supervised learning task. This is done by including porosity and specific surface area as inputs to the neural network along with the images. It was shown that this modification to the CNN provided better results than applying a regular CNN.

The same methodology of using a high-fidelity simulation applied to many porous media images to build an image-permeability dataset, then training a CNN was adopted in many applications involving porous media. The methodology was applied for gas diffusion layer materials [19], rock mechanics [20–22], and to fibrous porous media [23]. It must be noted that all the previously mentioned work deal with microscopic scale porous media images, whereas, in this paper, we develop our methodology for macroscale fibrous porous media images.

The previously mentioned techniques either offer an average of the permeability over the full macroscale domain or can do predictions on microstructures. Few research offers local permeability predictions on macrostructures such as the work of [24; 25]. In these works, airflow is used to flow in the porous media, in which the pressure profile is measured with sensors. The permeability related to the airflow is measured and correlated to the resin permeability. The research provides good results but will still need experimental work to get the data and to measure the correlation between air and resin permeability in new molds.

In this chapter, we provide a self-supervised learning framework to directly identify the permeability field of macro fibrous media based on 2D textile images, which can then be used to provide realistic accurate mold injection simulation. The framework is based on merging PINN and CNN together using data from central injection experiments. The developed technique is tested on unseen experiments and is seen to provide accurate results when comparing forward simulation based on the predicted permeability and existing experimental data.

The rest of the chapter is organized as follows: the framework of the central injection experiments and available data are presented in section 4.2. In section 4.3, the full computational methodology is explained which involves permeability labels generation using PINN, training a CNN, and the model evaluation methods. Section 4.4 provides testing results on unseen flow cases to assess the accuracy of the model. Section 4.5 offers a view of the possible future usage of the method which involves using it to make prediction for arbitrary 2D geometry and possibility of application to different fibrous media. Finally, section 4.6 offers a summary and a conclusion to the chapter.

4.2 Available data

4.2.1 Foreword

The present data were produced through 2D central injection experiments that were performed by Comas-Cardona at CACM (Centre for Advanced Composite Materials) at the University of Auckland (NZ).

4.2.2 Materials of the study

4.2.2.1 Liquid

The liquid used in this study is a mineral oil whose viscosity is 0.15 $Pa.s$ at 20^oC and has been measured on a Brookfield viscometer 4.1.

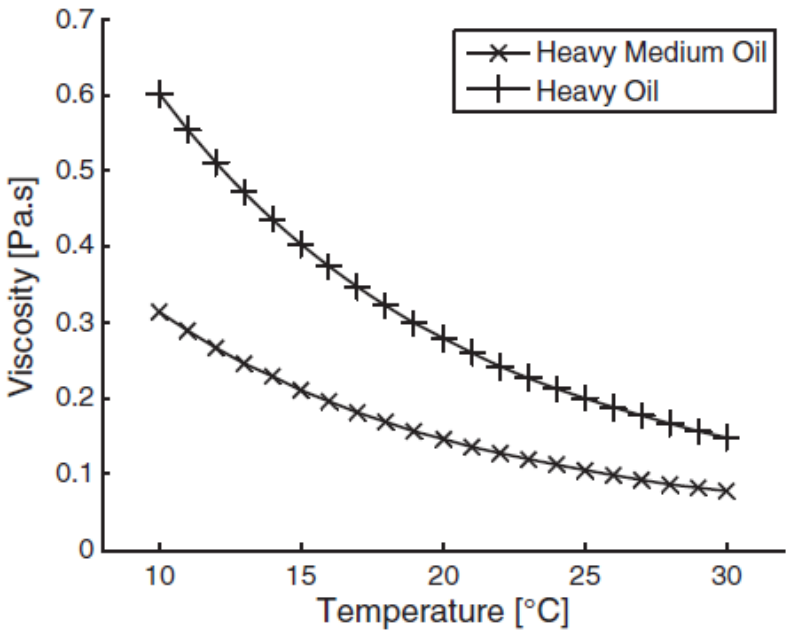

Figure 4.1: Viscosity measurements of the testing fluid [26].

4.2.2.2 Textile

The textile of interest is a glass fiber plain weave (PW) whose areal weight has been measured and is 822 g/m^2. It should be noted that the same fabric was used to produce the textile samples of all the experiments.

The average permeability of the textile was measured on 34 samples (single plies) following the procedure given by Swery et al. [26] and is shown in table 4.1.

Table 4.1: Average and standard deviation of the permeability of the plain weave fibrous media used in the central injection experiments. This average is calculated experimentally from 34 single ply samples.

	k_{xx} $(10^{-10}\ m^2)$	k_{yy} $(10^{-10}\ m^2)$	k_{xy} $(10^{-10}\ m^2)$
Average	0.69	3.28	0.077
Standard deviation	0.12	0.57	0.054

Before the injection starts, a lightbox image is taken of the dry fabric (2D plain weave) [27]. An example of the image is shown in figure 4.2. In the lightbox, light is transmitted through the textile. Light intensity is proportional to the areal weight of the textile. Two images are taken, the first one corresponds to the transmitted light without the textile and the second one corresponds to the transmitted light with the textile sample. Then, a difference is made pixel-wise between both images. Therefore, in white one can see the fiber tows at 0^o and 90^o of the weave, and in black the holes (opened areas) between the tows. The liquid resin will flow in the darker areas.

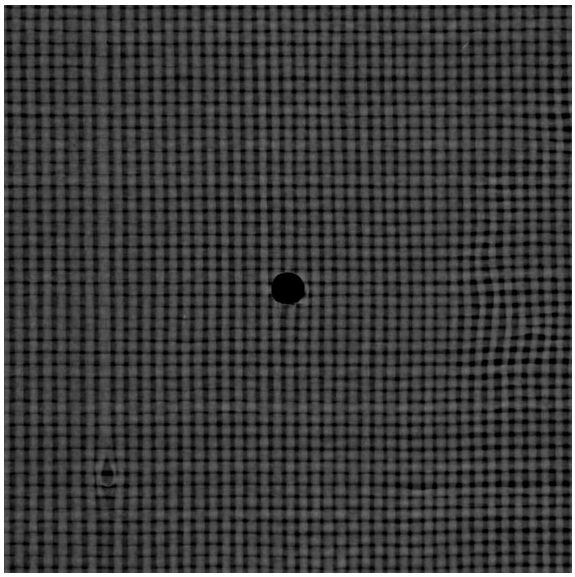

Figure 4.2: Lighbox image of a 2D plain weave textile sample used in one test.

The porosity of the tested textile is calculated for each sample using the mass of the ply m, the area of the ply A, the thickness of the ply h (measured as the cavity height between the compression platens), and glass density $\rho_f = 2.6\ g/cm^3$. According to the calculations, the samples have an average porosity of 0.45.

4.2.3 Injection Bench

The apparatus for the experiments is shown in figure 4.3 [26]. The apparatus consists of a fixed bottom transparent glass plate, a top moving stainless steel platen, a textile fabric, a tilted mirror, a camera, a pressure bucked, and an injection gate. The top and bottom platens are mounted into a universal testing machine. After closing the top platen on the textile fabric at a given thickness, the resin inlet valve

at the pressure bucket is opened. The resin will flow from the central injection point into the textile as the time goes on. For more details on the injection bench, the reader can refer to the work by Swery et al. [26].

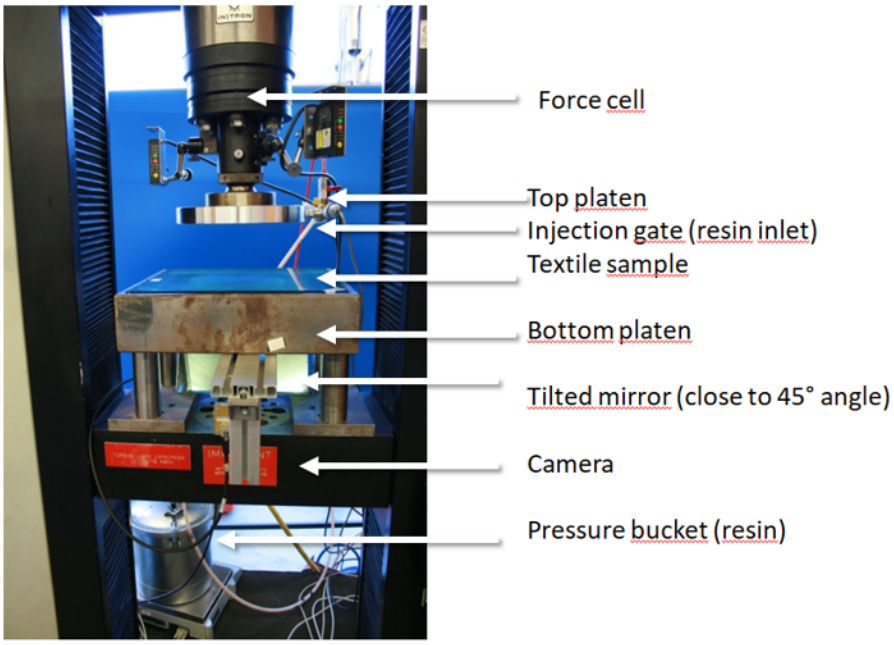

Figure 4.3: Apparatus used for the central injection experiments [26].

4.2.4 Monitoring and acquisition

The camera is used to take images of the flow at specified times to identify the location of the flow front. These raw images are saved and treated to produce black and white images which can then be used to extract the location of the interface that will be used for later analysis. An example of the raw images showing the flow front evolution is shown in figure 4.4. The calibration of the images has been done and is 0.3225 mm/px.

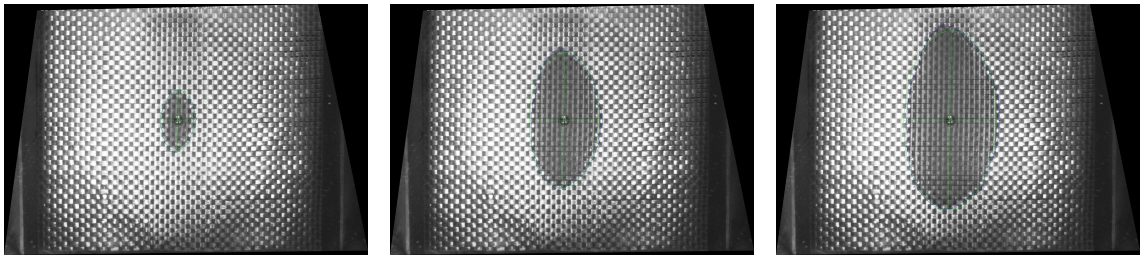

Figure 4.4: Raw flow front images at 3 different time instants of one of the central injection experiments. As time goes on, the liquid (grey central ellipse) flows into the porosity of the textile.

The treated segmented images are shown in figure 4.5.

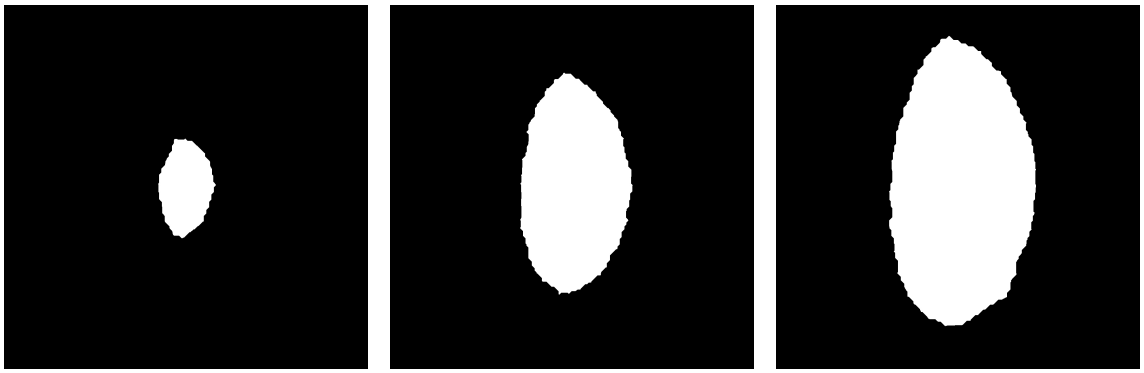

Figure 4.5: Treated flow front images at 3 different time instants of one of the central injection experiments, where the white part referred to resin impregnated textile and the black refers to dry textile.

A pressure sensor is located at the inlet that is used to measure the inlet pressure profile as a function of time. This data will also be used in the algorithm for permeability predictions. An example of the measured pressure vs. time is shown in figure 4.6.

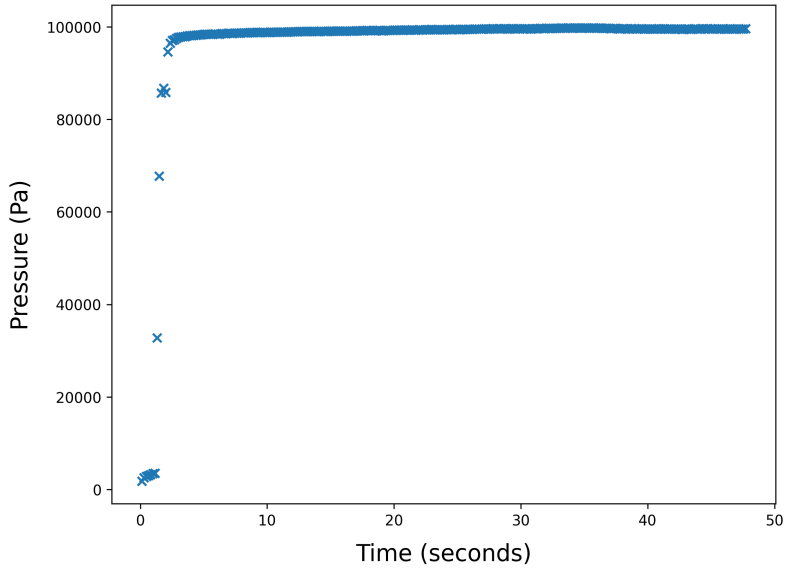

Figure 4.6: Data from pressure sensor showing pressure vs. time of one of the central injection experiments.

To summarize, we have data from 34 central injection experiments that includes

- flow front images with time,
- inlet pressure as a function of time,
- porosity and resin viscosity data,
- and a high-quality lightbox image of the dry textile (2D plain weave) taken prior to the injection.

4.3 Methodology

4.3.1 Neural network approximation of fields

The velocity $\mathbf{v}$, pressure p and resin volume fraction c fields for each experiments are approximated by feed forward neural networks having 5 hidden layers each with 20 units.

$$\mathbf{v}^{(j)} \approx \hat{\mathbf{v}}^{(j)}(x, y, t; \theta_v^{(j)}) \quad ; \quad p(j) \approx \hat{p}^{(j)}(x, y, t; \theta_p^{(j)}) \quad ; \quad c^{(j)} \approx \hat{c}^{(j)}(x, y, t; \theta_c^{(j)}) \tag{4.2}$$

where j denotes the experiment index, while $\theta_v^{(j)}$, $\theta_p^{(j)}$ and $\theta_c^{(j)}$ are the trainable parameters of the networks. We use *tanh* as activation function in all our networks. The output layer of $\hat{\mathbf{v}}^{(j)}$ has two units since the velocity field is a 2D vector field. Moreover, since the resin volume fraction c is naturally bounded between 0 and 1, we apply a sigmoid activation following the output layer of $\hat{c}^{(j)}$.

The permeability tensor is approximated using a CNN architecture:

$$\mathbf{K} \approx \hat{\mathbf{K}}(I; \theta_K) \tag{4.3}$$

Contrarily to velocity, pressure and resin fraction, the permeability model receives an $s \times s$ grayscale image $I \in \mathcal{R}^{s \times s}$ of the local textile configuration as an input and all the experiments share the same permeability model. The images I are obtained by cropping the original textile image $T \in R^{S \times S}$ at centers specified by collocation points. In our experiments $s = 50\ px$ and $S = 1150\ px$. The model outputs the three independent components of the 2D second order symmetric permeability tensor.

The CNN consists of convolutional, pooling, and fully-connected layers ending by the output, which is the permeability in our case. Batch normalization layers can also ensure that the inputs are well-normalized which improves the network performance. The trainable parameteres of the network are denoted by θ_K. The following CNN architecture is adopted:

- Input layer of images having a size of 50×50 pixels,
- A convolutional layer with 8 filters of size 3×3,
- A Batch Normalization layer,
- An Average pooling layer of size 2×2,
- A convolutional layer with 16 filters of size 5×5,
- An Average pooling layer of size 2×2,
- A fully connected layer with 64 neurons and tanh activation function,
- An output layer with 3 neurons corresponding to k_{xx}, k_{yy}, and k_{xy}.

4.3.2 Choice of the collocation points

Since the flow data come as a sequence of images, it is natural to select collocation points for residual evaluation of the PDEs at the centers of each pixel and at time frames given in the sequence. Therefore, we adopt the following notation:

$$x_l = (l - S/2) \cdot \Delta x \quad ; \quad y_m = (m - S/2) \cdot \Delta y \quad ; \quad t_n = n \cdot \Delta t \quad ; \tag{4.4}$$

where $\Delta x = \Delta y = 0.23mm$, $\Delta t = 2s$, $l, m = 0, 1, \ldots, S-1$ and $n = 0, 1, L-1$ ($L = 30$). We also introduce the global collocation index:

$$i = l + m \times S + n \times L \,. \tag{4.5}$$

Note that given a set (l, m, n) there is a unique corresponding global collocation index and vice versa, any i can be mapped back to a unique set (l, m, n). Using i allows to introduce the following naming convention:

$$x_i \,, y_i \,, t_i = x_l \,, y_m \,, t_n \,, \tag{4.6}$$

leading to a simplification in our notation, as we can write:

$$\hat{\mathbf{v}}_i^j = \hat{\mathbf{v}}^{(j)}(x_i, y_i, t_i, \theta_v^{(j)}) = \hat{\mathbf{v}}^{(j)}(x_l, y_m, t_n, \theta_v^{(j)}) \,. \tag{4.7}$$

Similarly, we can simplify the notation for $\hat{p}_i^j$ and $\hat{c}_i^j$, indicating the evaluation of the pressure and resin fraction models at collocation point i for the j-th experiment.

Finally, we indicate with

$$\hat{\mathbf{K}}_i^j = \hat{\mathbf{K}}(\mathsf{C}_s(T^j, x_l, y_m), \theta_K) \tag{4.8}$$

the permeability values obtained by evaluating the CNN model at collocation point i for the j-th experiment, where $\mathsf{C}_s(T^j, x_l, y_m)$ is the $s \times s$ cropping operator acting on the j-th textile image and centered at (x_l, y_m). It should be noted that a smaller random subset of the collocation points was used in practice.

4.3.3 Loss function

The loss function we seek to minimize includes both the physics loss from the governing equations and the data loss evaluated at a subset of N_{tc} collocation points across a number N_{te} of selected experiments for model training. A special treatment is due for the pressure boundary conditions as they must be enforced on a separate set of N_{bc} collocation points $(x_{\bar{i}}, x_{\bar{i}}, t_{\bar{i}})$ located on the domain boundaries. The loss can be formulated as follows:

$$\begin{aligned}\mathcal{L}(\theta_v^{(j)}, \theta_p^{(j)}, \theta_c^{(j)}, \theta_K) = \frac{1}{N_{te}} \sum_{j=0}^{N_{te}-1} (\lambda_c \, \mathcal{L}_c^j(\theta_c^{(j)}) + \lambda_p \, \mathcal{L}_p^j(\theta_p^{(j)}) + \lambda_1 \, \mathcal{L}_{f1}^j(\theta_v^{(j)}, \theta_p^{(j)}, \theta_K) \\ + \lambda_2 \, \mathcal{L}_{f2}^j(\theta_v^{(j)}) + \lambda_3 \, \mathcal{L}_{f3}^j(\theta_c^{(j)}, \theta_v^{(j)}))\end{aligned} \tag{4.9}$$

where

$$\mathcal{L}_c^j(\theta_c^{(j)}) = \frac{1}{N_{tc}} \sum_{i=0}^{N_{tc}-1} \left(c_i^j - \hat{c}_i^j\right)^2 \tag{4.10}$$

is the training mean squared error (mse) of the j-th resin volume fraction model;

$$\mathcal{L}_p^j(\theta_p^{(j)}) = \frac{1}{N_{bc}} \sum_{\bar{i}=0}^{N_{bc}-1} \left(p_{\bar{i}}^j - \hat{p}^{(j)}(x_{\bar{i}}, y_{\bar{i}}, t_{\bar{i}}; \theta_p^{(j)})\right)^2 \tag{4.11}$$

is the training mse of the j-th pressure model;

$$\mathcal{L}_{f1}^j(\theta_v^{(j)}, \theta_p^{(j)}, \theta_K) = \frac{1}{N_{tc}} \sum_{i=0}^{N_{tc}-1} \left(\hat{\mathbf{v}}_i^j + \frac{1}{\mu}\hat{\mathbf{K}}_i^j \cdot \nabla \hat{p}_i^j\right)^2 \tag{4.12}$$

is the training mean squared residual (msr) associated to the Darcy's law applied to the j-th model;

$$\mathcal{L}_{f2}^j(\theta_v^{(j)}) = \frac{1}{N_{tc}} \sum_{i=0}^{N_{tc}-1} \left(\nabla \cdot \hat{\mathbf{v}}_i^j\right)^2 \tag{4.13}$$

is the training msr associated with mass conservation of the j-th model and

$$\mathcal{L}_{f3}^j(\theta_c^{(j)}, \theta_v^{(j)}) = \frac{1}{N_{tc}} \sum_{i=0}^{N_{tc}-1} \left(\frac{\partial \hat{c}_i^j}{\partial t} + \hat{\mathbf{v}}_i^j \cdot \nabla \hat{c}_i^j \hat{\mathbf{v}}_i^j\right)^2 \tag{4.14}$$

is the training msr associated with the VOF transport equation applied to the j-th model. In our experiments we observed that any attempt to minimize the loss function in a monolithic way was leading to poor convergence regardless of the choice of the weighting coefficients λ. Therefore, we devised a segregated sequential training strategy detailed in the next subsection.

4.3.4 Sequential training

The point of departure of our training strategy is to replace the permeability model evaluations $\hat{\mathbf{K}}_i^j$ with as many trainable parameters $\tilde{\mathbf{K}}_i^j$. As a consequence of this choice, the loss function introduced in the previous section can be minimized over each experiment separately. Segregating flow experiments and training the corresponding networks separately leads to a remarkably superior convergence. Namely, for the j-th experiment, we denote the set of $\tilde{\mathbf{K}}_i^j$, $i = 0, 1, \ldots, \tilde{N}_{tc}$ trainable permeabilities as $\mathcal{K}^j$. In doing so, we consider only a fraction of the original collocation points (i.e. the pixel centers) in order to avoid overfitting due to introducing too many new trainable parameters. The modified loss functions can be rewritten as :

$$\begin{aligned}\mathcal{L}^j(\theta_v^{(j)}, \theta_p^{(j)}, \theta_c^{(j)}, \mathcal{K}^j) = \lambda_c\, \mathcal{L}_c^j(\theta_c^{(j)}) + \lambda_p\, \mathcal{L}_p^j(\theta_p^{(j)}) + \lambda_1\, \tilde{\mathcal{L}}_{f1}^j(\theta_v^{(j)}, \theta_p^{(j)}, \mathcal{K}^j) \\ + \lambda_2\, \mathcal{L}_{f2}^j(\theta_v^{(j)}) + \lambda_3\, \mathcal{L}_{f3}^j(\theta_c^{(j)}, \theta_v^{(j)})\end{aligned} \tag{4.15}$$

with the modified Darcy loss

$$\tilde{\mathcal{L}}_{f1}^j(\theta_v^{(j)}, \theta_p^{(j)}, \mathcal{K}^j) = \frac{1}{\tilde{N}_{tc}} \sum_{i=0}^{N_{tc}-1} \left(\hat{\mathbf{v}}_i^j + \frac{1}{\mu}\tilde{\mathbf{K}}_i^j \cdot \nabla \hat{p}_i^j\right)^2 . \tag{4.16}$$

Minimizing the loss function in equation (4.15) over the parameters $\theta_v^{(j)}$, $\theta_p^{(j)}$, $\theta_c^{(j)}$ and $\mathcal{K}^j$ is equivalent to solving an inverse problem using PINN to determine the permeability values. We can therefore gather these values from all the experiments

to create a data set consisting of image-label pairs $(I_i^j; \tilde{\mathbf{K}}_i^j)$ and use this to train the CNN model to predict permeabilities from local textile structures. A second and most important advantage offered by the sequential training strategy is that that we can apply proper filtering and augmentation techniques to this database leading to improved accuracy and generalization of the CNN model. Ultimately, the model is evaluated using an unseen flow experiment left out of the training set. For this new case, the permeability is predicted based from the evaluation of CNN model on the dry textile image, then PINN is used as a forward solver to compute the flow variables and finally the predicted filling pattern is compared to the experiments using the metrics defined in section 4.4. To summarize, the training strategy consists of the following steps:

1. Solving PINN for the identification inverse problem followed by data filtering and augmentation.
2. Training a CNN model to learn the mapping from the porous media images to the identified permeability.
3. Assessing model accuracy through comparison of predicted and experimental filling patterns for new test data.

The full methodology is summarized in figure 4.7.

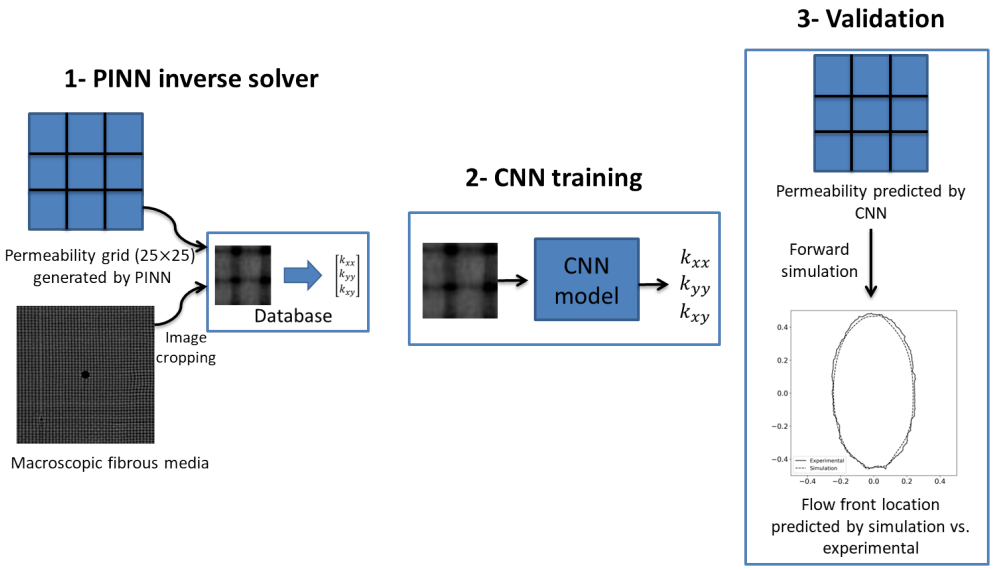

Figure 4.7: The full sequential training methodology to predict the permeability tensor field from textile images.

4.3.5 Identifying permeabilties using PINN

To solve the identification problem problem using PINN, different feedforward neural networks are used to approximate the pressure, fraction function, and the velocity. The computational domain is split into a grid of size 25×25, in which each subdomain will have an unknown permeability tensor, hence in this case $\tilde{N}_{tc} = 625$. These unknown permeabilities are treated similarly to the network parameters: after

initialization, values are updated using the minimization algorithm. The collocation points will have different permeabilities according to the subdomain they belong to. Adam optimizer is used for 5000 iterations which is followed by 500 BFGS iterations. Figure 4.8 shows the full PINN framework to solve the inverse problem.

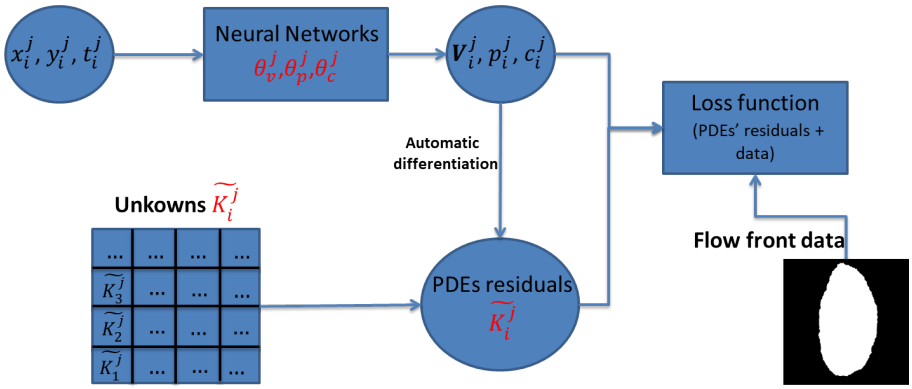

Figure 4.8: The framework to identify the permeability field by solving an inverse problem with PINN. The trainable parameters are identified in red.

An example of the identified permeability map is shown in figure 4.9 for one of the experiments.

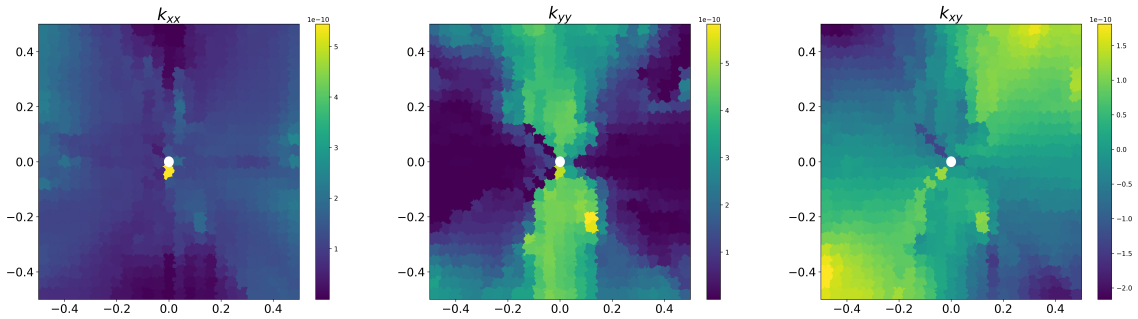

Figure 4.9: Example of the identified permeability field obtained by solving an inverse problem with PINN. The permeability component k_{xx}, k_{yy}, and k_{xy} are plotted from left to right, respectively. The x and y axes have been normalized; the real material size is $27 \times 27cm^2$

It is important to note that only a portion of the permeability field is utilized in generating these flow-front predictions, and there are certain regions within the field that cannot be considered trustworthy. Hence, a process of data cleaning is conducted, which will be elaborated upon and discussed in the subsequent section.

4.3.5.1 Permeability data cleaning

To enhance the reliability of the permeability data, a data cleaning procedure is implemented to eliminate information in which we have less confidence. One such type of data is associated with the porous media locations where the resin did not reach during the experiment. Consequently, the inferred permeability data in these regions cannot be considered valid. After excluding these unreliable data points, the resulting trustworthy areas are displayed in figure 4.10.

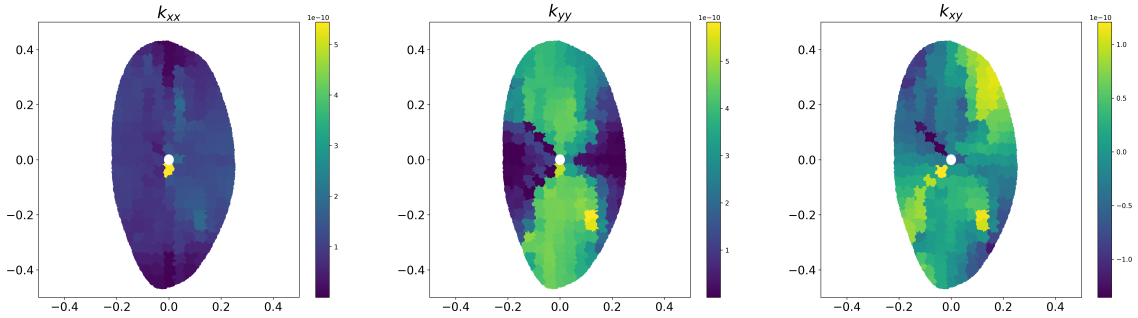

Figure 4.10: Example of the identified permeability field after trimming the regions where the resin did not reach. The permeability component k_{xx}, k_{yy}, and k_{xy} are plotted from left to right, respectively.

There is another category of data that is deemed untrustworthy, which pertains to the locations along the x and y axes. Specifically, in the case of k_{xx}, the data along the y-axis is considered unreliable since k_{xx} does not play a role in determining the flow in that direction. Similarly, for k_{yy}, the values along the x-axis are not trusted. The resulting version of the data that is considered trustworthy, after removing these unreliable values, is presented in figure 4.11.

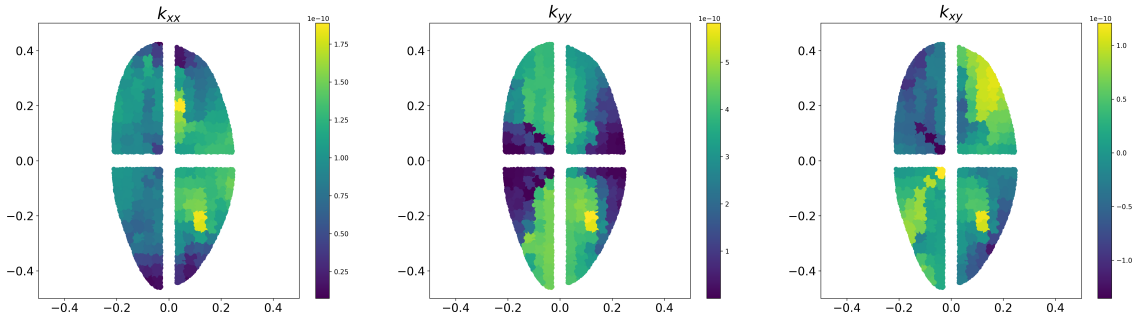

Figure 4.11: Example of the identified permeability field after all the trimming of the untrusted data. The permeability component k_{xx}, k_{yy}, and k_{xy} are plotted from left to right, respectively.

It should be noted that even though the data of k_{xx} along the x-axis and k_{yy} along the y-axis are trustworthy, in fact the most trustworthy in the data, they were removed so that each location have full permeability tensor information; so that we can use one convolutional neural network for the permeability tensor prediction.

4.3.5.2 Permeability data augmentation

The data that will be used in training the convolutional network is the lightbox images as in figure 4.2 and the cleaned permeability data as in figure 4.11. The lightbox images are cropped in 50×50 pixels images, where each crop corresponds to a permeability region. The data is, then, organized as a list of image crops and corresponding permeability tensors.

Data augmentation is used in order to increase the amount and variety of data used in the training. We mainly performed rotation of 90^o and 180^o on the cropped

images and obtained the corresponding rotated permeability tensor. The rotated permeability is obtained through the following equation:

$$\mathbf{K_{rot}} = \mathbf{R^T K R}, \tag{4.17}$$

where $\mathbf{R}$ is a rotation tensor that reads as:

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{4.18}$$

and θ is the angle of rotation.

The original dataset form along with the augmented data is shown in figure 4.12.

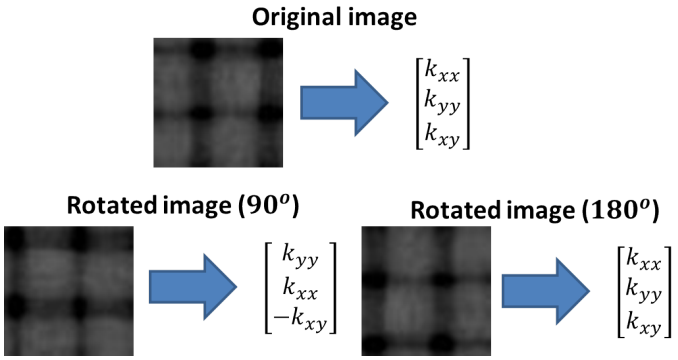

Figure 4.12: A sample of the data to be used for the convolutional neural network training which includes a lightbox cropped image and the corresponding permeability tensor labels. On top, one can find the cropped image and corresponding permeability tensor labels, and at the bottom, the augmented data through applying rotation of 90^o and 180^o along with rotated permeability tensor data.

4.3.6 Training the CNN permeability model

The CNN is trained from scratch on the cleaned and augmented dataset obtained from the previous stage. The dataset size obtained from 33 experiments (one experiment is left out for testing purposes) is about $18,000$. This is split into training and validation sets with a ratio of $9:1$. Mean absolute error is used as the loss function to be minimized. Adam minimization algorithm is used for 1000 epochs with an initial stepsize of 0.001, with batch size of 128. In the training process, convergence is determined upon observing a divergence between the training and validation errors for over 100 consecutive iterations. The model associated with the minimal validation error is subsequently retained for testing.

The loss function over the training and validation sets are shown in figure 4.13.

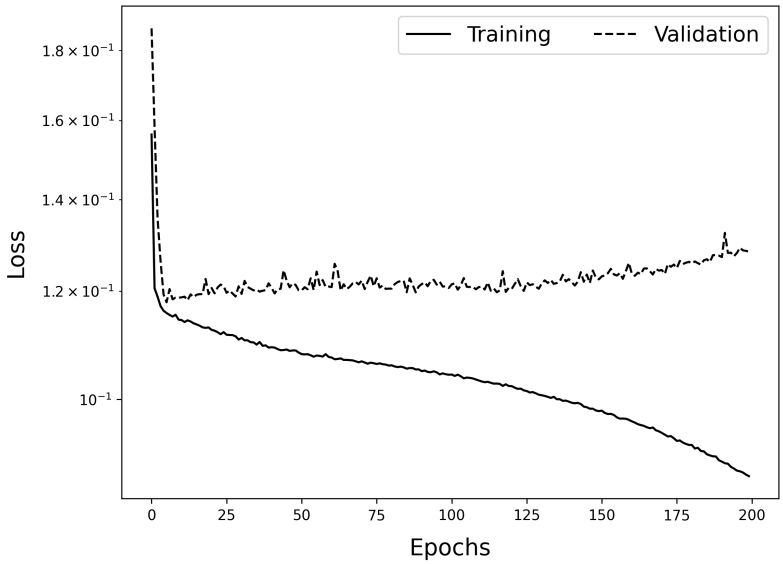

Figure 4.13: Loss function evolution vs. the number of epochs for the training and the validation data sets without applying regularization.

As can be seen from figure 4.13, the model tends to overfit the training data which is shown in the validation loss which tends to increase with the number of epochs. L_2 regularization is used to solve this issue. The weight of the regularization term is set to 0.001. The loss function is plotted after the regularization in figure 4.14 which shows that L_2 regularization solved the overfitting problem.

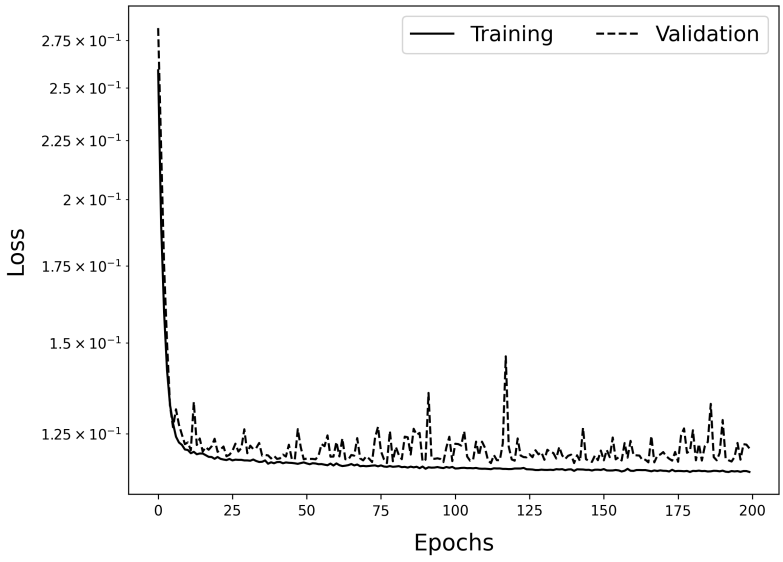

Figure 4.14: Loss function evolution vs. the number of epochs for the training and the validation data sets.

4.3.7 Model evaluation

In order to perform testing on unseen data we use cross-validation. Each one of the 34 experiments is left out each time and the training procedure is started using the remaining 33 experiments. After the model is trained, it is used to predict the

permeability of the left-out test. The predicted permeability is used to perform forward simulation and the results are compared to the existing experimental flow front images. This process is repeated for all 34 tests, where the CNN is retrained from scratch. The results of the simulation using the CNN-predicted permeability field are also compared with the simulation using an average experimental measure of the permeability.

Two different error measures are used to assess the performance of the model. The first is the absolute difference between the experimental flow front image at the final time step and the simulation results which measures the percentage of the mispredicted pixels, referred to as e_p and defined as:

$$e_p = \frac{\int_\Omega 2|c - c_{exp}| \; d\mathbf{X}}{\int_\Omega c \; d\mathbf{X} + \int_\Omega c_{exp} \; d\mathbf{X}}, \tag{4.19}$$

where c is the fraction function solution using the permeability prediction method and c_{exp} is the fraction function from the experiment.

The second error uses the Hausdorff distance [28] which is a measure of the distance between two shapes (set of points), in our case the predicted flow front shape compared to the experimental flow front shape. The Hausdorff distance between two sets X and Y is defined as:

$$d_{\mathrm{H}}(X, Y) = \max \left\{ \sup_{x \in X}(\inf_{y \in Y} d(x, y)), \sup_{y \in Y}(\inf_{x \in X} d(x, y)) \right\}, \tag{4.20}$$

where sup represents the supremum, inf the infimum, and where $d(x, y)$ quantifies the distance from a point $x \in X$ to a point $y \in Y$. Figure 4.15 can help to visually understand the two error definitions given two curves (flow front shapes), X and Y.

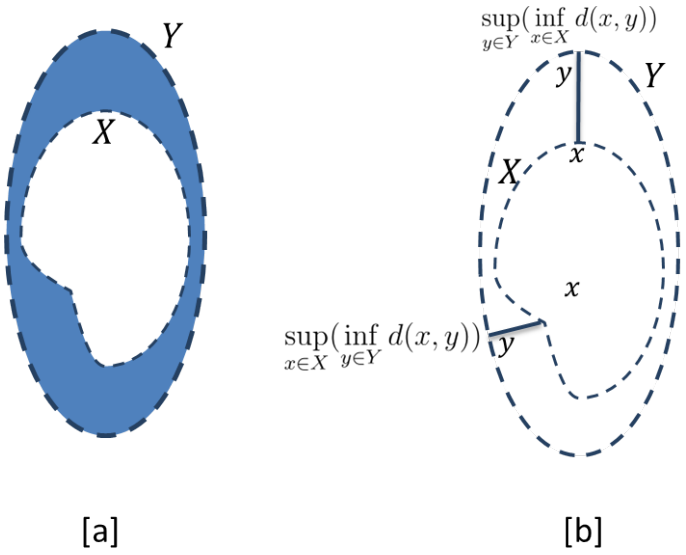

Figure 4.15: [a] A graphical representational of the error e_p, where the shaded area between the two flow front shapes, X and Y, represents the mispredicted pixels. [b] The Hausdorff distance between two curves, X and Y [28].

Following the definition of the Hausdorff distance, the second error measure, referred to as e_d, can be defined as:

$$e_d = \frac{H_d}{H_0} \tag{4.21}$$

where H_d is the Hausdorff distance between the model and experimental flow front, and H_0 is the Hausdorff distance from the experimental flow front shape to its center of mass which is used for normalization.

4.4 Results

The two error metrics distributions are plotted for all 34 tests in histograms (figures 4.16 and 4.17). The y-axis represents the number of occurrences and the x-axis represents the error in percentage.

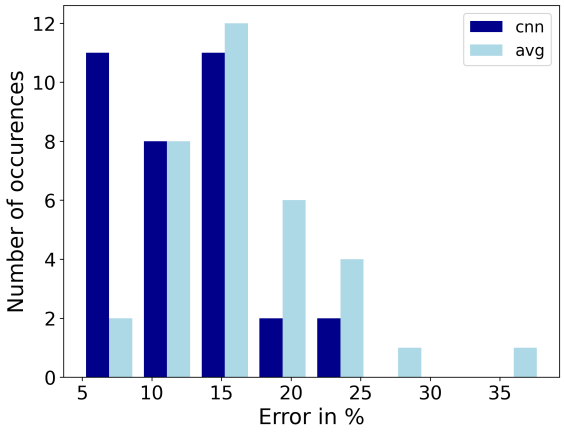

Figure 4.16: Histogram plot showing the number of occurrences in the y-axis and the mispredicted pixels error e_p measure in percentage in the x-axis for the simulation using CNN-predicted permeability and the simulation using the average permeability.

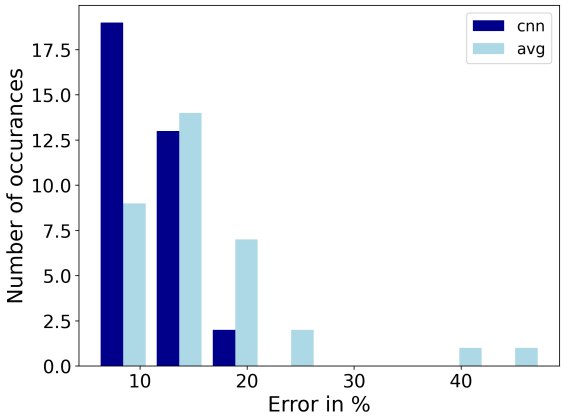

Figure 4.17: Histogram plot showing the number of occurrences in the y-axis and the shape error e_d measure in percentage in the x-axis for the simulation using CNN-predicted permeability and the simulation using the average permeability.

It can be generally seen from figures 4.16 and 4.17 that using CNN-predicted permeability field produces less error than using an average approximation to the permeability. That is seen as the bars related to the CNN are more shifted to the left towards the low error region. From figure 4.16, 11 tests using CNN predictions produced near 5% error, while only 2 tests using the average permeability values produced this level of accuracy. Moreover, using average values, some tests produced error near 30% and 35%. However, none of the tests using CNN produced such high errors. For the second error measure in figure 4.17, all the tests using the CNN are near the 10% error range except for 2 tests near the 20% range. However, results using the average permeability values are more scattered with 7 tests near the 20% range and some tests in the range of $30 - 40\%$ error range.

In general, the central injection of plain-weave fabrics results in an anisotropic elliptical flow behavior according to experiments. In some of the test cases, the experimental injection front has a shape close to an ellipse, meaning that there are low local defects or variability. In that case, both the CNN-predicted permeability and the average permeability produced comparable results as in test case number 21 plotted in figure 4.18. In this test, the CNN produced an error of 6.6%, while the average permeability produced an error of 12.1% using the mispredicted pixels error. Using the shape error, the CNN produced an error of 5.8%, while the average permeability produced an error of 9.0%.

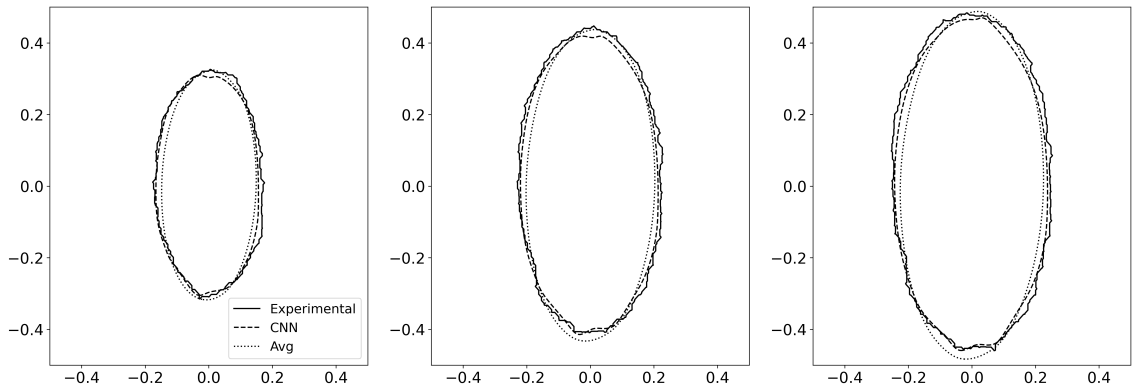

Figure 4.18: Flow front positions at three different time instants for the experimental, simulation using the CNN-predicted permeability, and simulation using average permeability for test number 21.

If there is high local variability or local defects in the textile sample, it is expected that the flow will not follow an elliptical shape. We show two test cases where the experimental flow front is has more of a circular behaviour (less anisotropic). It could be due to local effects such as gaps or misorientation between tows in the samples of fabric used. The two tests, numbers 24 and 26, are plotted in figures 4.19 and 4.20, simultaneously. The CNN managed to accurately capture these effects from the images producing a good prediction for the local permeability which was proven by the accurate flow simulation that matched the experimental flow front shape. As opposed to the results using an average permeability which, by essence, could not capture this effect.

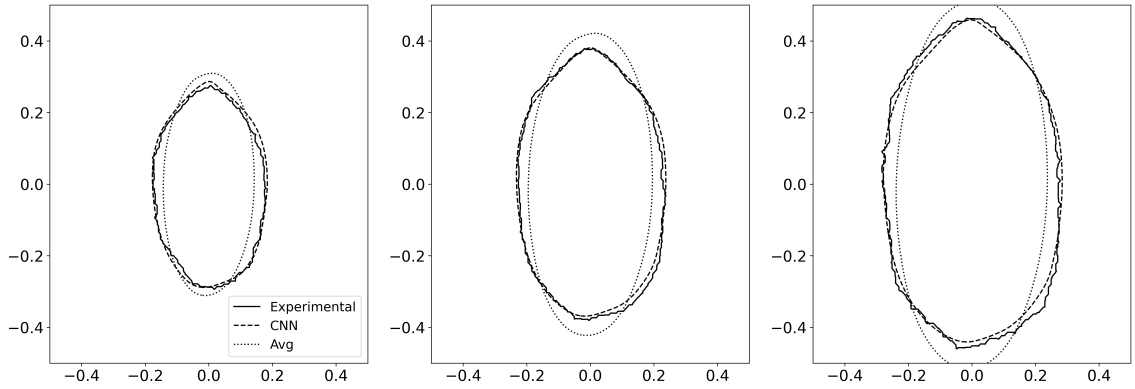

Figure 4.19: Flow front location at 3 different time instants of experiments number 24. The figures show the experimental flow front along with the simulated flow front using the CNN predicted permeability and the measured average permeability value.

For test number 24 (figure 4.19), the CNN produced an error of 5.3%, while the average permeability produced an error of 20.5% using the mispredicted pixels error. Using the shape error, the CNN produced an error of 7.4%, while the average permeability produced an error of 16.0%.

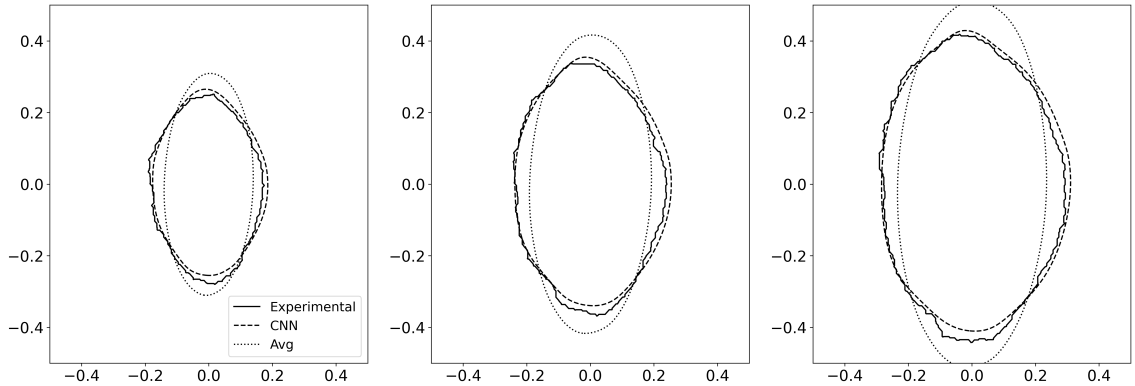

Figure 4.20: Flow front location at 3 different time instants of experiments number 26. The figures show the experimental flow front along with the simulated flow front using the CNN predicted permeability and the measured average permeability value.

For test number 26 (figure 4.20), the CNN produced an error of 6.9%, while the average permeability produced an error of 26.2% using the mispredicted pixels error. Using the shape error, the CNN produced an error of 9.2%, while the average permeability produced an error of 24.2%.

4.5 Perspectives

The presented method offers a way to obtain an approximation of the permeability field from textile images directly. There are two main points to discuss about the limitations and possibilities of the method. The first is the possibility of generalizing the technique to predict the permeability for any 2D geometry and problem other

than the central injection experiment. The second is applying the technique to a different fibrous structure other than the plain weave structure which was used in the data generation and model training.

4.5.1 Prediction for arbitrary geometries

The primary advantage and strength of our technique lie in establishing a connection between the macroscopic geometry image and the localized small image crops. Through training the CNN on these image crops, our model gains the ability to predict the permeability field for any 2D geometry.

The image crops, irrespective of the macroscopic geometry, exhibit consistent characteristics. This consistency enables the prediction of the permeability field for an arbitrary 2D planar geometry, making it independent of the specific geometric configuration. In essence, our method is local and therefore "geometry agnostic," meaning it can potentially handle diverse geometries without requiring specific adaptations or modifications. This consideration deserves further investigation to find definitive confirmation.

The method presented in this work might also be extended with some due adaptations to 3D geometries. For thin shell-like structures the method could be applicable as it is, provided that the 2D flow images can be mapped back to the 3D geometry. As for bulk 3D parts, flow and dry fibers imaging only offer a view that is limited to the external surfaces, while data for the inner part is not available.

4.5.2 Generalization for different fibrous media

The training of the CNN involved utilizing data obtained from experiments conducted on a plain weave matrix structure. To enhance the dataset and improve the model's ability to generalize, data augmentation techniques were employed. However, a crucial question remains regarding the model's performance when applied to different fibrous media structures such as twill or honeycomb weaves. Answering this question conclusively necessitates conducting experiments specifically designed to test the model's capabilities.

Although it is challenging to provide a definitive answer without experimental validation, preliminary tests suggest that the model may possess a certain level of generalization, at least qualitatively, to different fibrous structures. However, it is important to note that the model's predictions may not necessarily be quantitatively accurate for these structures.

It would be valuable in the future to expand the training set by incorporating data from various fibrous structures. By training the model on different fibrous media structures, the generalizability of the model can be enhanced. This broader training approach would allow the model to learn and adapt to the unique behaviors and features present in different fibrous structures, potentially improving its predictive

performance.

Moreover, it should be noted that the transmitted light images, as the ones used in this work, only work for transparent (glass) fibers, and will not work for carbon or natural fibers which are opaque to visible light.

4.6 Summary and Conclusion

This chapter offers a methodology to perform accurate realistic flow simulation in fibrous porous media through having image-based estimation of the permeability field produced by a trained convolutional neural network. The chapter can be summarized in the following points:

- Variations in fibrous porous media take place from one sample to another and also spatially within the same sample. These variations can lead to unpredicated flow behaviour or defects in some cases. To consider these variations and have accurate predictions from simulations, it is essential to treat the permeability as a field not a single average value, and this field should be different from one sample to another.

- To this end, we built a framework using PINN and CNN to obtain the permeability field from macroscale fibrous media images. The method starts with permeability field data collection by solving an inverse problem with PINN utilizing experimental flow front images. This framework allowed us to collect data of the permeability fields from 34 central injection experiments.

- We developed a CNN model to directly predict the permeability field from macroscale images of fibrous media. The model was trained using crops of the macroscale image correlating the crops to the corresponding local permeability. By performing the training on image crops, the method possesses the ability to be used for any 2D geometry. Through cross-validation, we demonstrated that the CNN model possesses generalizability and can provide a reliable approximation of the permeability tensor for any 2D plain weave lightbox image. It should be emphasized that once the model is trained, the model predictions are made only from a dry textile image without the need for any injection data.

The work presented in this chapter is novel in the sense that it relates a macroscale image to the corresponding permeability field. To the best of my knowledge, there is no published work that offers a similar strategy that can be applied to fibrous media images.

To conclude, PINN was successfully used to solve an inverse problem to identify the permeability field using data from flow front images and an inlet pressure sensor. This framework is used to collect data on the permeability fields of 34 central injection experiments.

The collected data and the corresponding lightbox images are used to train a CNN to learn the relationship between the image crops of the macroscale fibrous media image and the permeability tensor field. Cross-validation showed that the CNN model is general and can be used for performing accurate flow simulations using PINN or any existing solver which is a step towards realistic flow simulation in fibrous porous media.

The main originalities in this chapter are

- the PINN framework to solve an inverse problem to identify the permeability field,
- and correlating crops of macroscale fibrous media to the corresponding permeability tensor.

4.7 Bibliography

[1] Phillip C Carman. Fluid flow through granular beds. *Chemical Engineering Research and Design*, 75:S32–S48, 1997.

[2] T G Gutowski, Z Cai, S Bauer, D Boucher, J Kingery, and S Wineman. Consolidation experiments for laminate composites. *Journal of Composite Materials*, 21(7):650–669, 1987.

[3] Timothy G Gutowski, Tadahiko Morigaki, and Zhong Cai. The consolidation of laminate composites. *Journal of Composite Materials*, 21(2):172–188, 1987.

[4] B Rikard Gebart. Permeability of unidirectional reinforcements for rtm. *Journal of composite materials*, 26(8):1100–1133, 1992.

[5] Sanjay Sharma and Dennis A Siginer. Permeability measurement methods in porous media of fiber reinforced composites. *Applied Mechanics Reviews*, 63(2), 2010.

[6] K Ken Han, C William Lee, and Brian P Rice. Measurements of the permeability of fiber preforms and applications. *Composites science and Technology*, 60(12-13):2435–2441, 2000.

[7] YJ Lee, JH Wu, Y Hsu, and CH Chung. A prediction method on in-plane permeability of mat/roving fibers laminates in vacuum assisted resin transfer molding. *Polymer composites*, 27(6):665–670, 2006.

[8] KL Adams, WB Russel, and LJIJoMF Rebenfeld. Radial penetration of a viscous liquid into a planar anisotropic porous medium. *International Journal of Multiphase Flow*, 14(2):203–215, 1988.

[9] JR Weitzenböck, RA Shenoi, and PA Wilson. Radial flow permeability measurement. part a: Theory. *Composites Part A: Applied Science and Manufacturing*, 30(6):781–796, 1999.

[10] Shakil Ahmed and Stefan Iglauer. Brine permeability predictions for sand packs and sandstones using navier-stokes equations and three-dimensional micro-tomography images of pore spaces. In *9th International conference on CFD in the Minerals and Process Industries, Melbourne, Australia*, 2012.

[11] Naoki Takano, Masaru Zako, Toru Okazaki, and Kenjiro Terada. Microstructure-based evaluation of the influence of woven architecture on permeability by asymptotic homogenization theory. *Composites science and technology*, 62(10-11):1347–1356, 2002.

[12] Gerd Morren, Massimo Bottiglieri, Sven Bossuyt, Hugo Sol, David Lecompte, Bart Verleye, and Stepan V Lomov. A reference specimen for permeability measurements of fibrous reinforcements for rtm. *Composites Part A: Applied Science and Manufacturing*, 40(3):244–250, 2009.

[13] Aydin Nabovati, Edward W Llewellin, and Antonio CM Sousa. A general model for the permeability of fibrous porous media based on fluid flow simulations using the lattice boltzmann method. *Composites Part A: Applied Science and Manufacturing*, 40(6-7):860–869, 2009.

[14] Hyunjun Cho, Namgyun Jeong, and Hyung Jin Sung. Permeability of microscale fibrous porous media using the lattice boltzmann method. *International Journal of heat and fluid flow*, 44:435–443, 2013.

[15] Nattavadee Srisutthiyakorn*. Deep-learning methods for predicting permeability from 2d/3d binary-segmented images. In *SEG technical program expanded abstracts 2016*, pages 3042–3046. Society of Exploration Geophysicists, 2016.

[16] Krzysztof M Graczyk and Maciej Matyka. Predicting porosity, permeability, and tortuosity of porous media from images by deep learning. *Scientific reports*, 10(1):21488, 2020.

[17] Haiyi Wu, Wen-Zhen Fang, Qinjun Kang, Wen-Quan Tao, and Rui Qiao. Predicting effective diffusivity of porous media from images by deep learning. *Scientific reports*, 9(1):20387, 2019.

[18] Jinlong Wu, Xiaolong Yin, and Heng Xiao. Seeing permeability from images: fast prediction with convolutional neural networks. *Science bulletin*, 63(18):1215–1222, 2018.

[19] Taylr Cawte and Aimy Bazylak. A 3d convolutional neural network accurately predicts the permeability of gas diffusion layer materials directly from image data. *Current Opinion in Electrochemistry*, page 101101, 2022.

[20] Pengfei Tang, Dongxiao Zhang, and Heng Li. Predicting permeability from 3d rock images based on cnn with physical information. *Journal of Hydrology*, 606:127473, 2022.

[21] Mohamed Elmorsy, Wael El-Dakhakhni, and Benzhong Zhao. Generalizable permeability prediction of digital porous media via a novel multi-scale 3d convolutional neural network. *Water Resources Research*, 58(3):e2021WR031454, 2022.

[22] Stephan Gärttner, Faruk O Alpak, Andreas Meier, Nadja Ray, and Florian Frank. Estimating permeability of 3d micro-ct images by physics-informed cnns based on dns. *Computational Geosciences*, 27(2):245–262, 2023.

[23] Baris Caglar, Guillaume Broggi, Muhammad A Ali, Laurent Orgéas, and Véronique Michaud. Deep learning accelerated prediction of the permeability of fibrous microstructures. *Composites Part A: Applied Science and Manufacturing*, 158:106973, 2022.

[24] Li Ding, Chiang Shih, Zhiyong Liang, Chuck Zhang, and Ben Wang. In situ measurement and monitoring of whole-field permeability profile of fiber preform for liquid composite molding processes. *Composites Part A: Applied Science and Manufacturing*, 34(8):779–789, 2003.

[25] Xugang Ye, Jingjing Qiu, Chuck Zhang, Richard Liang, and Ben Wang. A finite element-based heuristic estimation of local preform permeability for resin transfer molding. *Transport in porous media*, 76:247–263, 2009.

[26] Elinor E Swery, Tom Allen, Sebastien Comas-Cardona, Quentin Govignon, Chris Hickey, Jamie Timms, Loic Tournier, Andrew Walbran, Piaras Kelly, and Simon Bickerton. Efficient experimental characterisation of the permeability of fibrous textiles. *Journal of composite materials*, 50(28):4023–4038, 2016.

[27] JM Gan, S Bickerton, and M Battley. Quantifying variability within glass fibre reinforcements using an automated optical method. *Composites Part A: Applied Science and Manufacturing*, 43(8):1169–1176, 2012.

[28] William Rucklidge. *Efficient visual recognition using the Hausdorff distance*. Springer, 1996.

Chapter 5

Summary, Conclusion, and Perspectives

Abstract

The final chapter of the thesis is dedicated to summarizing the main contributions and drawing conclusions. A section discussing the perspectives and recommended future work is added at the end of the chapter. The perspectives are split into future work related to PINN and future work related to composite manufacturing.

Contents

5.1 Summary

Three main contributions were presented in this thesis which are

- building a PDE solver for two-phase flow problems in porous media using PINN,
- assessing the use of PINN to build metamodels in such problems,
- building a framework to identify the permeability field of 2D fibrous media from images.

A quick summary of each contribution is summarized in the next sections.

5.1.1 PINN as a PDE solver

Chapter 2 of the thesis focuses on the application PINN to solve forward problems associated with two-phase flow in porous media. The method has been effectively employed to solve problems in both one-dimensional (1D) and two-dimensional (2D) settings. Importantly, the framework developed in this chapter can be easily adapted to address any 2D problem by modifying the collocation points, as well as the initial and boundary conditions.

To enhance the accuracy of the solutions provided by PINN, an adaptivity technique has been introduced. This technique is particularly crucial for addressing the presence of a moving discontinuity, such as a flow front, within the solved problems. The adaptivity algorithm developed in the study is not limited to specific scenarios but is instead applicable to solve a wide range of partial differential equations (PDEs) of interest.

5.1.2 Building Metamodels with PINN

Chapter 3 of the thesis examines the utilization of PINN for constructing metamodels to applied two-phase flow in porous media problems. The chapter presents a range of examples, both in 1D and 2D scenarios. PINN proves to be a promising approach for such metamodeling tasks, particularly when compared to traditional methods and model order reduction techniques that struggle to handle the presence of discontinuities in the solution [1–3].

Furthermore, the chapter introduces a more cost-effective method for building metamodels, specifically designed for sensitivity analysis applications. This novel approach is referred to as "SA-PINN," which stands for Sensitivity Analysis-PINN. By leveraging the SA-PINN method, the process of constructing metamodels becomes more affordable, addressing the economic constraints associated with sensitivity analysis tasks.

5.1.3 Permeability field identification from images

In Chapter 4, a comprehensive framework was developed to tackle the challenging task of identifying the permeability field from 2D macroscale fibrous media images. The methodology devised for this purpose begins by solving an inverse problem using PINN and incorporating experimental data to collect permeability field information. This combination of PINN and experimental data serves as the basis for generating a dataset.

To leverage the collected permeability data and corresponding fibrous media images, a supervised learning approach employing CNN is used. The objective is to train the CNN to learn a function that effectively maps an input image to its corresponding permeability tensor, thus establishing a relationship between the visual characteristics of the fibrous media and the permeability properties.

To validate the accuracy and performance of the CNN predictions, unseen test cases are utilized. The CNN predicts the permeability based on the input image, which is then utilized in forward simulations. The output of these simulations is subsequently compared to existing experimental data, allowing for a quantitative assessment of the CNN's performance and the reliability of the permeability predictions.

5.2 Conclusion

The main goals of the thesis as discussed in the introduction are to improve the:

1. robustness of LCM processes through online control.
2. material characterization before the process.

To serve the first end, steps have been taken to assess the use of PINN to build surrogate models which is a first step towards online control. It has been shown that PINN is able to build such models even for our complicated model with a moving discontinuity which is quite a complicated task using other methods such as model order reduction. However, more work is still needed in the future to build such models for large-scale applications and to include a control agent which can be done using reinforcement learning for example.

For the second goal, a technique to accurately predict the permeability field, an important property of the porous media which is highly variable from one part to another, tailored to each specific part is developed. The developed technique showed great results in its ability to make accurate simulations matching the experimental results whereas classical methods that offer an average of the permeability over the whole domain cannot accurately make such predictions. The new technique is geometry agnostic since it can be applied to different 2D geometries.

Throughout the thesis, developments in PINN have been made to improve its approximation capabilities. For example, a new technique to adapt the collocation points based on the residual field was developed and tested on 1D and 2D problems.

This shows that PINN can be used accurately as a PDE solver for two-phase flow in porous media problems.

In summary, this thesis has showcased the versatility and efficacy of PINN in solving forward problems, developing metamodels, and identifying permeability fields. The introduction of an adaptivity technique, the flexibility of the framework, and the successful integration with CNNs underscore the potential of PINN as a powerful tool in the field of two-phase flow in porous media. These findings pave the way for further advancements and applications of PINN in solving complex problems across various scientific and engineering domains.

5.3 Perspectives and Future Work

Numerous avenues can be explored in future research to enhance the hyperparameter tuning, accuracy, and computational efficiency of Physics-Informed Neural Networks (PINN). Additionally, there are specific directions that hold direct relevance to the field of composites manufacturing. This section is further divided into two categories: future work pertaining to PINN and future work pertaining to composites manufacturing.

5.3.1 PINN related future work

Various aspects of PINN need attention in future investigations. Improvements can be made in hyperparameter tuning techniques to optimize the network architecture and training parameters for enhanced performance. We will focus on three main aspects which are the multi-objective optimization process, the minimization algorithm, and parallelization of the algorithms.

5.3.1.1 Multi-objective optimization

The training process of PINN involves minimizing a loss function using an optimization algorithm. This loss function comprises multiple terms, each with different scales, which compete with each other to be minimized. This process can be categorized as multi-objective optimization, as there are several objectives to be simultaneously minimized.

These terms in the loss function represent the residual of the partial differential equation (PDE), the enforcement of initial and boundary conditions, and the incorporation of available data. In the case of coupled problems, such as those addressed in this thesis, multiple PDEs and conditions must be considered. Consequently, the number of terms in the loss function can be substantial, as demonstrated in Chapter 4, where we encounter 9 terms in the loss function. This increased number of terms adds complexity to the optimization process, posing additional challenges.

The first recommended step, which was carried out in this thesis, involved non-dimensionalizing the entire system of equations [4]. This process ensures that all terms in the equations have comparable magnitudes, reducing the disparity between them

and simplifying the optimization process. However, even with non-dimensionalization, further adjustments may be required since different terms may have different acceptable error thresholds. Thus, weighting the terms becomes necessary.

In the thesis, most of the examples involved the weighting of different terms. This weighting process was primarily accomplished through trial and error, which can be a tedious and laborious task to determine the optimal weights for a specific problem.

Although various methods from the literature were explored to automate the weight-tuning process, they proved effective only for simpler problems with two or three objectives [5–7]. When applied to the complex coupled problem addressed in the thesis, these methods did not yield satisfactory results.

However, there is a promising strategy that shows potential for resolving this issue: the strict enforcement of initial and boundary conditions. Several research papers have addressed this strategy, which involves selecting an approximate solution, denoted as $\hat{u}$, that automatically satisfies the imposed conditions [8; 9]. An example of this approximation is given as follows:

$$\hat{u}(x,t) = u_{NN}(x,t)D(x,t) + u_p, \tag{5.1}$$

where u_{NN} is a neural network approximation, D is a distance function that takes a value of 0 on the boundaries and grows in the domain, and u_p is a function that exactly satisfies the conditions to be enforced.

By utilizing this approximation, the enforcement of conditions becomes exact, as $\hat{u}(x,t) = u_p$ at the boundary. However, a significant challenge arises when dealing with the exact distance function D, as it can introduce instabilities and complicate the optimization process. Furthermore, obtaining an accurate distance function can be challenging for complex geometries and conditions.

To overcome these challenges, an alternative approach is to find a suitable approximation for the distance function D. This research direction has gained attention in some promising papers and presents a potential solution for future investigations [10; 11]. Developing an effective approximation for the distance function would address the issues associated with instabilities and the complexity of getting an exact D, while also enabling the application of this methodology to complex geometries.

5.3.1.2 Minimization algorithm

In the PINN community, it has become common practice to use a powerful first-order minimization algorithm, typically Adam, initially in the training process to reach the general vicinity of the global minimum followed by a second-order minimization algorithm, such as BFGS or l-BFGS, to quickly approach the minimum more closely. However, this two-step minimization process is not optimal due to certain drawbacks.

One issue with the BFGS optimizer is its computational expense and inefficient memory usage, as it requires constructing an approximation of the inverse of the Hessian matrix. Although l-BFGS addresses the memory problem and is faster than BFGS, it does not achieve the same level of minimization as BFGS.

Furthermore, using BFGS and l-BFGS directly in TensorFlow is not straightforward, requiring the use of another library like TensorFlow Probability. This implementation is not efficient and may contain bugs, such as producing a non-positive definite approximation of the Hessian matrix, which is theoretically incorrect. In contrast, PyTorch offers a simple and user-friendly l-BFGS implementation but lacks support for BFGS.

An alternative algorithm that shows promise is the Levenberg-Marquardt algorithm, which has been found to be more efficient and robust than BFGS or l-BFGS while achieving similar accuracy as BFGS [12; 13]. However, direct implementations of this algorithm in TensorFlow or PyTorch do not currently exist, but other libraries can be explored and tested.

Overall, further research is necessary to explore advanced optimization algorithms specifically tailored for PINN which could lead to more efficient and effective training processes. It is also crucial to understand the specific requirements of each problem to choose the most appropriate algorithm.

5.3.1.3 Parallelization

In this context, it is important to note that all the algorithms discussed were originally designed to run on either a single CPU or GPU. The choice of algorithm does not depend on whether a CPU or GPU machine is used. However, the aspect of parallelization was not considered during the development of these algorithms.

For future advancements, it is worth modifying the code to leverage multiple GPUs. By utilizing parallel processing, it is possible to significantly accelerate the training process beyond its current speed.

5.3.2 Composite manufacturing future work

Several topics can be addressed as future research related to the manufacturing process of study in this thesis, resin transfer molding. One topic can be the extension to 3D and thin structures. Another is adding heat transfer and polymerization physics to the coupled system of equations.

5.3.2.1 3D and thin structures extension

Extending the algorithms to 3D problems is a straightforward process, requiring no additional concepts or difficulties beyond the increased computational cost. Although the thesis did not specifically address the 3D extension, there are no new scientific concepts to explore; however, there is significant industrial interest in this area.

When dealing with 3D model's training, there is a need for 3D injection experimental data. Significant experimental work remains to be developed and performed so as to extract valuable data. Several non destructive techniques (X-ray tomography or dielectric for instance can provide bulk steady or transient information but are not easy to build and will require investigation to provide data.

Of greater interest and potential impact is the reshaping of the equations and codes to handle 3D thin structures, which are commonly encountered in LCM processes. Since most LCM-manufactured parts consist of thin structures, commercial finite element codes typically employ shell elements. Therefore, adapting the equations and codes to accommodate such structures would be a valuable contribution, particularly within the composite manufacturing community.

To initiate this extension, a paper discussing the application of PINN to thin structures in the context of solid mechanics applications can serve as a starting point. By adopting similar concepts and methodologies, the equations pertinent to the current research can be adapted to handle 3D thin structures, presenting an intriguing and potentially impactful research direction.

5.3.2.2 Multi-Physics extension

LCM processes involve the simultaneous occurrence of flow in porous media, polymerization, and heat transfer within the mold. Consequently, there is a need for a simulation tool capable of solving this complex multi-physics system as a whole.

In the future, it would be beneficial to explore the application of PINN to tackle this intricate multi-physics problem in LCM. The main challenge lies in dealing with the optimization of multiple objectives. Even with an approach to enforce conditions the hard way, there will still be the task of minimizing residuals from multiple PDEs simultaneously. Consequently, the development of an automated weight-choosing algorithm becomes crucial in such cases.

5.4 Bibliography

[1] David Neron and Pierre Ladeveze. Idelsohns benchmark. Technical report, Technical report, 2013.

[2] Liqian Peng and Kamran Mohseni. Nonlinear model reduction via a locally weighted pod method. *International Journal for Numerical Methods in Engineetming*, 106(5):372–396, 2016.

[3] Andrea Ferrero, Angelo Iollo, and Francesco Larocca. Global and local pod models for the prediction of compressible flows with dg methods. *International Journal for Numerical Methods in Engineering*, 116(5):332–357, 2018.

[4] Ryno Laubscher and Pieter Rousseau. Application of mixed-variable physics-informed neural networks to solve normalised momentum and energy transport

equations for 2d internal convective flow. *arXiv preprint arXiv:2105.10889*, 2021.

[5] Mohannad Elhamod, Jie Bu, Christopher Singh, Matthew Redell, Abantika Ghosh, Viktor Podolskiy, Wei-Cheng Lee, and Anuj Karpatne. Cophy-pgnn: Learning physics-guided neural networks with competing loss functions for solving eigenvalue problems. *ACM Transactions on Intelligent Systems and Technology*, 13(6):1–23, 2022.

[6] Colby L Wight and Jia Zhao. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020.

[7] Remco van der Meer, Cornelis W Oosterlee, and Anastasia Borovykh. Optimally weighted loss functions for solving pdes with neural networks. *Journal of Computational and Applied Mathematics*, 405:113887, 2022.

[8] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[9] Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

[10] N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

[11] S Berrone, C Canuto, M Pintore, and N Sukumar. Enforcing dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks. *arXiv preprint arXiv:2210.14795*, 2022.

[12] John Taylor, Wenyi Wang, Biswajit Bala, and Tomasz Bednarz. Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. *arXiv preprint arXiv:2205.07430*, 2022.

[13] Gaurav Kumar Yadav and Balaji Srinivasan. Shallow physics informed neural networks using levenberg-marquardt optimization.

Titre : Modélisation de l'écoulement diphasique dans les milieux poreux à l'aide de réseaux neuronaux informés par la physique pour des applications dans le moulage de composites par injection de résine

Mots clés : métamodèles, prédiction de la perméabilité à partir d'images, réseaux neuronaux convolutionnels, apprentissage auto-supervisé

Résumé : Le moulage de composites liquides (LCM) est une famille populaire de procédés de fabrication de composites, dans lesquels une résine liquide est injectée dans un moule où un textile est positionné. Le procédé implique un écoulement dans un milieu poreux fibreux. Les variations au sein des échantillons textiles et entre eux peuvent être dues à des défauts géométriques intrinsèques du tissu, à une mauvaise manipulation, à un mauvais alignement et à d'autres facteurs. Ces incohérences peuvent entraîner des écarts marqués entre les modèles de remplissage réels et prédits, ce qui entraîne des variations dans la qualité des pièces fabriquées. Cette thèse a deux objectifs principaux. Le premier est de construire un cadre qui prédit la possibilité d'apparition de défauts d'injection, ce qui peut faciliter le processus de prise de décisions correctives. Le second est d'améliorer la caractérisation des propriétés clés des matériaux avant le début de l'injection. Pour réaliser ces tâches, nous utilisons des réseaux neuronaux informés par la physique (PINN). PINN repose sur la fusion de la connaissance des données et la physique, représentée par des équations aux dérivées partielles. Pour atteindre le premier objectif, PINN est utilisé pour construire des métamodèles du processus avec des paramètres d'intérêt tels que la perméabilité ou les conditions limites d'entrée. Ces modèles sont formés hors ligne et peuvent être rapidement utilisés pour les prédictions en ligne. Pour atteindre le deuxième objectif, un cadre d'apprentissage auto-supervisé a été construit sur la base de PINN et de réseaux neuronaux convolutionnels pour identifier le champ tensoriel de perméabilité à partir d'images textiles en 2D. Le cadre montre des résultats prometteurs en les comparant aux images expérimentales existantes du front d'écoulement.

Title : Modeling two-phase flow in porous media using physics-informed neural networks for applications in liquid composite molding

Keywords : metamodels, Image-based permeability prediction, convolutional neural networks, self-supervised learning

Abstract: Liquid composite molding (LCM) is a popular family of composite manufacturing processes, in which a liquid resin is injected in a mold where a textile is set there. The process involves flow in fibrous porous media. Variabilities within and between textile samples can arise due to fabric intrinsic geometrical defects, mishandling, misalignment, and other factors. These inconsistencies can result in marked deviations between the actual and anticipated filling patterns, leading to variations in the manufactured parts quality. This thesis has two main objectives. The first is to build an online framework that predicts the possibility of defects, which helps taking corrective decisions. The second is to improve the characterization of key material properties before the injection starts. To achieve these tasks, we use physics-informed neural networks (PINN). PINN is a technique that is based on merging the data knowledge along with the knowledge of physics, represented by partial differential equations. To target the first objective, PINN is used to build metamodels of the process with parameters of interest as the permeability or inlet boundary conditions. These models are trained offline and can be quickly employed for online predictions. Towards the second objective, a self-supervised learning framework was built based on PINN and convolutional neural networks to identify the permeability tensor field from 2D textile images. The framework shows promising results through comparing with existing experimental flow front images.