Solving Physical Problems using ML: Application to Defects in Alloys

Submitted in partial fulfilment of the requirements for the degree of Master of Technology

By

Shivprasad Kathane

Roll No. 180110076

Under the guidance of
Prof. Shyamprasad Karagadde

Centre for Machine Intelligence and Data Science

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

October 2022

Declaration

I declare that this written submission represents my ideas based on my understanding and I have adequately cited and referenced the original source where others' ideas or words have been included. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and can evoke penal action from the sources which have thus not been properly cited or from whom the proper submission has not been taken when needed.

Shivprasad Kathane
180110076

Acknowledgement

I would like to take this opportunity to express gratitude to my guide, Prof. Shyamprasad Karagadde for guiding me in this project, especially with his suggestions on what possible aspects I can work upon and the steps I can take as part of my research. Meetings with him, updating on my work and receiving his inputs have instilled a sense of confidence in me. I also thank all the professors of Indian Institute of Technology, Bombay especially from Metallurgical Engineering and Material Science and Machine Intelligence and Data Science, who have imparted me foundational knowledge and instilled in me interest to pursue research. I am also thankful to my friends and family for their support and encouragement.

Shivprasad Kathane

IITB, 16 October 2022

Abstract

Prediction of defects in metal alloys is very critical in engineering applications to avoid failure and establish quality control. While physical models are conventionally used to capture process-structure-property relationships with simulations used to monitor behaviour under different conditions, the rise of deep learning and imaging data provides opportunity to build highly accurate data-driven models for prediction even in unknown scenarios. However, rather than utilising them purely as 'black-box' models, it is desired to embed them with process-related information and make them more explainable and effective. Physics-Informed Neural Network (PINN) is a recent framework in this regard and this report delves into it. A literature review is presented along with illustrations of how PINNs are used to solve partial differential equations while learning from data and constraining to conditions. Design and implementation of PINN is discussed and an experiment to interpret the correlation between the tuned PINN design and the PDE it models is presented. It is expected that a combined data-driven and physics-informed model will be ideal for defect prediction.

Table of Contents

1. Introduction & Outline **6**

2. Literature Review **7**
- 2.1. ML for Microstructural Studies 7
- 2.2. Defect Detection using DL 8
- 2.3. Prediction of Microstructural Evolution 13
- 2.4. Physics-Informed Neural Networks 16
- 2.5. Examples of Problems Solved 21
- 2.6. Summary & Objectives 24

3. Implementation & Results **25**
- 3.1. Solving an ODE 25
- 3.2. Solving a PDE using PINN 27
- 3.3. Predicting Evolution of Solution 30
- 3.4. Linear Convection Diffusion PDE 32
- 3.5. PINN Design & Implementation 34

4. Summary & Conclusion **38**
- 4.1. Advantages of PINNs 38
- 4.2. Challenges 39
- 4.3. Future Steps 39

5. References **40**

1. Introduction & Outline

Metal alloys form critical components of a lot of engineering structures such as in automobiles and aircrafts where component failure can be even fatal. The failure mainly arises due to defects present in the alloy, hence it becomes important to establish proper defect control procedures. Based on the scale (dimension) at which they are present, these defects can be classified as point (atomic size or 0D), line (1D), surface (2D), and volume (3D) defects. Examples include interstitials, vacancies, dislocations, precipitates, grain boundaries (tilt and twist), twin boundaries, disclinations, and even cracks. These defects might arise during various processes such as forming, casting, welding, and additive manufacturing. These processes can often be modelled using numerical simulations if the differential equations representing them are known. However, in the absence of a well-determined equation governing the system, any available information should be utilised for analysis. This includes recorded experimental (numerical) data, known conditions or constraints for the system, information on material properties, process parameter data, and visual (image) data. For example, a defect might occur when say process parameters such as the cooling rate or material parameters such as the composition deviate to undesired values. A data-driven model can capture such a correlation but it would be too general. For a desired specific application, it needs to be infused with constraints and physical laws relevant to the process. This is where the concept of a physics-informed model finds utility.

The structure of material when observed at the microscopic scale revealing the spatial organisation of grains, grain boundaries, impurities, and defects is called the microstructure. Imaging systems such as computed tomography (CT) have been used to record microstructure images. This project aims to use existing 3D microscopic data on large metal alloy components, and develop physics-informed, data-driven predictive models for identifying locations of defects. In order to build such a model, it requires a thorough review of the literature familiarising with existing methods and understanding of recently established frameworks. This report begins with the applications of ML in microstructural studies with a focus on using deep learning for defect detection. It then moves on to prediction of microstructural evolution with the idea of learning physics from images. Then, a detailed understanding of Physics-Informed Neural Networks (PINNs) is presented. Examples of different problems solved in previous studies are included. Thereafter, all the experiments performed are described and the obtained results are illustrated. This includes solving a simple ODE using a neural network, solving PDEs using PINNs, delving into the design and implementation aspects of PINN for better interpretation and hyperparameter tuning. Next, a reflection on the literature review and results (summary) is presented noting the significance of PINNs in general and also towards the overall objective of research. Finally, the report ends with a discussion on the challenges and future steps.

2. Literature Review

2.1. ML for Microstructural Studies

There have been several instances of the utilisation of machine learning and deep learning for microstructural studies in materials science. When it comes to microstructures, which are recorded in the form of images, it is popular to use convolutional neural networks (CNN) which auto-extract image features and then also perform the learning task. [3] mentions the application of CNN algorithms for various research purposes including prediction of mechanical properties, steel microstructure classification, microstructure reconstruction and structure-property prediction, steel microstructure segmentation, casting defect detection in X-ray images, microstructure-property explorations in photovoltaics, etc. The next section specifically discusses defect detection using deep learning.

In contrast to CNN, the traditional image-processing research has thrived on synthesising features explicitly based on expertise and then applying machine learning techniques to perform the required task. This means some form of understanding and analysis of the domain is captured and the models built are explainable but are less automated and accurate as compared to deep learning.

[5] illustrates the use of ML and computer vision methods for automatic microstructure recognition. It performs 2 classification tasks namely, detecting the presence of dendritic morphologies in micrographs and identifying the type of cross-sectional view. This involved feature extraction using techniques like visual bag of words, texture and shape statistics, and pre-trained convolutional neural networks followed by dimensionality reduction to obtain a relevant reduced feature set. Finally, classification was performed using support vector machines, nearest neighbours, and random forest models. It was found that the use of pre-trained neural networks to extract features followed by classification by SVM yielded the best results with accuracies being greater than 90% for both tasks. See Fig 1.

Fig 1: Approach for ML-based classification tasks on microstructural image data [5]

In contrast to the above supervised classification task, [6] introduces an unsupervised ML-based technique for autonomous microstructural characterization in 3D samples. It combines topology classification, image processing, and clustering algorithms and unbiasedly performs quantification of grains and their size distributions and characterises features such as voids and porosity: things that are physically interpretable. Thus, this technique fulfils the purpose of efficiently identifying, tracking, and quantifying complex microstructural features that have been known to impact and explain the observed material behaviour. See Fig 2 and Fig 3.

Fig 2: Unsupervised ML workflow for autonomous microstructural characterization [6]

Fig 3: Identifying grains and their size distribution in polycrystalline samples of Al & Fe [6]

2.2. Defect Detection using DL

Most of the literature relevant to the problem statement illustrates approaches for defect detection from image data. Electron microscopy has been widely used to study defects in crystal structures. But automating the analysis of images and detection of defects is necessary in order to establish a fast, accurate and reliable process that can be scaled up and useful to apply in real-time. This is where machine learning, deep learning, and computer vision based approaches have

gained attention. These largely include the segmentation of images using deep learning to classify the type of defect present in the image.

[1] investigates the problem of finding the location and determining the geometry of different defect clusters in irradiated steels. It demonstrates a deep learning based Faster R-CNN analysis system which obtained a performance comparable to human analysis with limited data. This system consists of 2 modules (Fig 4). First is the Faster Region-based Convolutional Neural Network (F R-CNN) Detector which uses convolutional layers to extract a feature map from an input image. The map is acted upon by the Region Proposal Network which places the predefined bounding boxes and identifies the specific Regions of Interest. Each of these is acted upon by a classifier to determine the defect type and a regressor for getting a refined location. The next module utilises geometry fitting algorithms to determine the shape and size of each defect identified by the earlier module. The model achieved upto 83% recall.

Fig 4: Defect Detection: Modules A) Faster R-CNN Detector B) Image Property Analysis [1]

Fig 5: Defect Detection: Prediction vs Ground Truth for dislocations in an EM image [1]

Similarly [2] demonstrates semantic segmentation of defects in transmission electron microscopy (TEM) images of steel. A CNN-based DefectSegNet architecture (connected encoder-decoder with multiple convolutional and pooling layers) was introduced to segment 3 types of defects present in the images: dislocation lines, precipitates, and voids. It achieved greater than 90% accuracy across all the 3 types and outperformed the human expert average. See Fig 6 and Fig 7.

Fig 6: Illustration of the DefectSegNet architecture [2]

Fig 7: Defect Detection: Comparing Prediction & Ground Truth for dislocations in TEMIs [2]

[3] illustrates the use of a CNN model to identify casting defects (porosities) in optical microscope images of different aluminium alloy systems with varying magnifications. The model achieved a classification accuracy of 94%. See Fig 8 and Fig 9.

Fig 8: A CNN architecture directly employed on microstructure image patches [3]

Fig 9: Defect Detection a) marked by human expert b) predicted by DL model [3]

There could be many more instances of such models for defect detection but essentially these act as black-box models and information on how the model works is mostly unavailable or the basis comprises experience, heuristics, and intuition something which makes model explainability difficult in deep learning research. Mostly, the design of experiments technique of identifying various model hyperparameters and the range of values they can take up is adopted and optimization is performed to obtain the right set of values that yield the highest performance. There is a need to analyse the best network in order to effectively understand the problem-solution match. This might include interpreting what each layer of the neural network captures. In this direction, [4] performs quantitative manufacturing defects (micrometre scale) detection experiments based on the

AlexNet (a deep CNN) architecture on microscopy images of Ti-6Al-4V. It investigates a variety of design parameters whereby the optimal one achieves an average accuracy of 98.54%. It uses the notion of entropy derived from information theory to quantify the information gain through the layers of the network. The mutual information between the filter and label is established at each layer in order to compare the discriminating (defect vs no defect) power of features which was found to be highest at the last layer. See Fig 10 and Fig 11.

Fig 10: Mutual information analysis reveals filter i has higher discriminating power than j [4]

Fig 11: Mutual Information distribution across layers (L) and Detection Probability Map (R) [4]

However, what most of these approaches do not take into account is the knowledge of the processes or the underlying phenomena which the images don't reveal by themselves. Even though these approaches might work in real-time they evaluate each image on a standalone basis ignoring the larger perspective of what's happening in the material or system in order to effectively predict future evolution. Hence, this motivates the need for an approach to learn the physics from image data or imbibing prior process knowledge while designing the networks in order to better explain the (otherwise black-box) models built and make appropriate predictions.

2.3. Prediction of Microstructural Evolution

In order to better understand the processing-structure-property relationships, studying how the microstructure evolves with time is crucial. Traditionally, microstructure evolution has been modelled using simulations with governing principles described by partial differential equations (PDEs) behind the process.

[7] demonstrates that convolutional recurrent neural networks can learn the underlying physical rules and can replace the PDE-based simulations in the prediction of microstructure phenomena with increased efficiency. The RNN used here was Eidetic 3D LSTM which applies convolution to both the spatial (2) and temporal (1) dimensions resulting in integrated spatiotemporal feature extraction capable of both good-enough long-term and short-term predictions. The learning ability and predictive power of the network were assessed for four types of phenomena in order of increasing complexity: plane-wave propagation, grain growth, spinodal decomposition, and dendritic crystal growth. The training was performed by self-supervised learning from image sequences generated by the numerical simulation of these processes. It was ensured that this phase reflects the practical scenario of the availability of incomplete information about the systems or underlying mechanisms by not supplying material parameters and using not well-determined PDEs for the simulation. Yet, the network was able to produce a reliable prediction of image sequences with microstructures representing accurate short-term local dynamics and long-term statistical properties of the material-process system.

Fig 12: Conv-RNN combines convolution which generates latent representations with RNN which is responsible for the sequential (temporal) evolution of these representations [7]

Here is a glimpse of their approach for plane-wave propagation and dendrite growth:

The dynamics of scalar field c representing the plane-wave propagation is given by:

$$c\,(x,\ y,\ t)\ =\ [\tfrac{1}{2} sin\,(k_x x\ +\ k_y y\ +\ \omega t\ +\ \theta)\ exp\,(-\ \beta t)]\ +\ \tfrac{1}{2} \tag{1}$$

where k = (k_x, k_y) is the wave vector, θ is a random phase, and β is a decay exponent

Equation 1 was used to generate the image sequences and the RNN was used to predict the next 50 frames when supplied with 10 consecutive input frames.

Fig 13: RNN Predicted vs Ground Truth output frames for plane-wave propagation [7]

Pixel-wise comparison of the prediction with ground truth was performed by calculating RMSE and SSIM. The parameters present in the equation were extracted from the predicted images in order to gauge the extent to which the RNN is able to learn the underlying evolution rules. The resultant metrics were good-enough.

Fig 14: Left = Root Mean Square Error (RMSE) and Structural Similarity Index Measure (SSIM) between predictions and ground truths as a function of frame j. Right = Relative errors of wave-propagation parameters (k, ω, β) inferred from the predicted images [7]

Phase-field model was used to simulate the dendritic solidification process. The time evolution of coupled φ and T fields representing the process is given by:

$$\tau\partial\phi/\partial t\ =\ -\ \partial F/\partial\phi \tag{2}$$

$$\partial T/\partial t\ =\ \nabla^2 T\ +\ K\partial\phi/\partial t \tag{3}$$

where T = temperature, ϕ = order parameter, F = free energy, K = latent heat

The coupled equations 2 and 3 were used to generate the image sequences and as before the RNN was used to predict the next 50 frames when supplied with 10 consecutive input frames. In Fig 15, input frames corresponding to the parameter K = 1.161 were supplied to the RNN. K was kept hidden from the learning process yet the RNN was able to predict the correct evolution trajectory based on input frames.

Fig 15: RNN Predicted vs Ground Truth output frames for dendritic growth [7]

Pixel-wise comparison of the prediction with ground truth was performed by calculating RMSE and SSIM. Shape descriptors, namely: ferret diameter, convexity, and solidity of the growing crystal were calculated from the predicted (P) and ground truth (G) images and compared. The resultant metrics were good-enough.

Fig 16: A) RMSE and SSIM between predictions and ground truths as a function of frame j
B) Time evolution of ferret diameter, convexity, and solidity of a growing crystal [7]

This study goes a notch above the other deep learning research studies by successfully demonstrating that the DL model is able to learn the underlying physics by identifying characteristic features from the set of input image data and thus is able to explain the high-accurate predictions of time-evolution of microstructure images. However, still, it makes use of numerical simulations for generating input images which is time-consuming and the model varies with the underlying parameters.

2.4. Physics-Informed Neural Networks

Physics-Informed Neural Networks are neural networks that take into account information from the differential equation reflecting the physics of the problem within the learning process. It is a scientific machine-learning technique used to solve problems involving partial differential equations (PDEs). A need for such kind of framework arose mainly because standard numerical approaches tend to be time-consuming or too difficult to solve some PDEs that represent significant non-linearities, convection dominance, or shocks and also to take advantage of the learning power of neural networks. PINNs represent a multi-task learning framework with the dual objective of fitting the observed data and minimising the PDE residual.

The idea of PINN originates from the universal function approximation capabilities of neural networks because of which they can be used to model any function, say u, dependent on inputs, say x (space) and t (time). If NN is the neural net output, then:

$$NN \approx u\,(x,\ t) \tag{4}$$

Now, if we have a differential equation in u then neural networks can be used to solve it if we can iteratively learn to model u while simultaneously obeying the physical law represented by the equation. This idea can be traced back to [8] which presented a method to solve initial and boundary value problems using artificial neural networks. It proposed to start with a trial solution of the differential equation as a sum of two parts, one satisfying the known conditions and containing no adjustable parameters (A) while the other uses a feedforward neural network (with adjustable weights) to satisfy the differential equation without affecting the conditions (F).

Let the differential equation be represented by G = 0. Here G will be a function of input x, the solution to be computed Ψ and the derivatives of Ψ (upto order n say 2):

$$G\,(x,\ \psi(x),\ \nabla\psi(x),\ \nabla^2\psi(x)) = 0 \quad x \in D,\ the\ domain \tag{5}$$

If a neural network with weights p is used to model ψ, then we train the network (i.e. vary p) to satisfy Eq. 5 subject to the given conditions. The trial solution is written as:

$$\psi(x) = A(x) + F(x,\ NN(x,\ p)) \tag{6}$$

Here, by construction, ψ satisfies the conditions given in A while the objective of neural network training is to minimise the sum of G values over data points x.

This approach was general enough to be applicable to single ordinary differential equations (ODEs), systems of coupled ODEs, and also to PDEs although a separate trial function needs to be constructed depending on the equation.

Similarly, PINNs approximate PDE solutions by training a neural network to minimise a loss function which includes terms reflecting the initial conditions (at start time) and boundary conditions (in space), and the PDE residual at selected points in the domain (collocation data). Traditionally, neural networks learn solely from data by minimising the error between predictions and labels but PINNs are novel in the sense that they make the neural network learn to respect any symmetries, invariances, or conservation principles by integrating the mathematical model into the network by designing a custom loss function. The PDE residual acts as a penalising term in the loss function and restricts the space of acceptable solutions. The PINN training is an unsupervised mesh-free strategy that does not require results from prior simulations or experiments (necessary for a general supervised deep learning problem) and is free from step limitations (that otherwise arise in numerical simulations). [9]

There have been an increasing number of papers on PINNs addressing problems of increasing complexities over the years starting from the 1D Schrodinger's Equation (SE) in 2018, followed by Euler Equation (EE) to model high-speed aerodynamic flows, incompressible Navier-Stokes Equation (NSE) and the coupled system of NSE and temperature equation to analyse heat flow convection (NSE+HE) in 2021.

Fig 17: History snapshot of problems solved using PINN in literature [9]

PINNs were first proposed in [10] where the framework was demonstrated for solving classical problems in fluids, reaction–diffusion systems, quantum mechanics and nonlinear shallow-water waves propagation.

Parametrized and nonlinear partial differential equations were considered to be of the form:

$$u_t + \aleph[u;\lambda] = 0, \quad x \in \Omega, \quad t \in [0, T] \tag{7}$$

where u (t, x) as before denotes the latent (to be determined) solution and $\aleph$ denotes a non-linear operator parameterized by λ, while $\Omega \subset \Re^d$ is the domain.

Two distinct problems were of interest when provided with noisy measurements of the system. Determining the hidden state u (t, x) of the system given fixed model parameters λ (corresponding to the PDE) formed the forward problem of inference, filtering, and smoothing, or data-driven solution of PDE. The inverse problem was of determining the parameters λ that best explain the observed data which is also known as learning, system identification, or data-driven discovery of PDE. [10] showcased how to build continuous-time models in the PINN framework for solving Schrodinger's Equation in the forward case and Navier-Stokes Equation in the inverse scenario.

Define $f = u_t + \aleph[u;\lambda]$ (8)

Then, approximating u (t, x) by a deep neural network results in the physics-informed neural network f (t, x). f can be derived by applying chain rule on the composition of functions using automatic differentiation. It has the same parameters as u but due to the action of $\aleph$ has extra final activation functions.

Fig 18: A traditional neural network with 2 neurons in the input layer corresponding to inputs I_1 and I_2, 2 neurons in the hidden layer with weights W1 & W2 and biases b1 & b2, and 1 neuron in the output layer corresponding to O expressed as a function of W, b, I_1, and I_2. [11]

A traditional neural network such as the one shown above is used to train over initial (t = 0) and boundary ($\partial\Omega_D$) data points (collection of inputs I_1 = x & I_2 = t and output label O = u). This represents a supervised learning setup.

Fig 19: An illustration showing the parts of input space used for training PINN [11]

The collocation input data (points within the domain or on the boundary) after passing through the traditional neural network yield corresponding outputs which along with their differentiations wrt the inputs and PDE parameters $\lambda = \Theta$ are used to evaluate the PDE as $f = \Pi$. This forms a physics-informed neural network which represents an unsupervised learning setup (no label for collocation data).

Fig 20: The forward problem is identifying u. This requires evaluating PDE value Π (system output) given inputs I_1 and I_2, system parameter Θ, and neural network output O (approximate of u). This value can be expressed as γ being a function of W, b, I_1, I_2, and Θ. Estimating the unknown parameter set Θ given input-output data is the inverse problem. [11]

The PINN training proceeds by minimising the loss function L wrt parameters p:

$$L_p(X) = L_p^{\,o}(X^o) + L_p^{\,b}(X^b) + L_p^{\,r}(X^r) \tag{9}$$

where X denotes the collection of training data

This is a non-convex loss function often minimised using gradient-based optimization.

The loss term consists of three mean square terms:

1. Mean Squared Residual over Collocation Data $X^r = \{(t_i^r, x_i^r)$ for i = 1 to $N_r\}$

$$L_p^r(X^r) = 1/N_r * \sum_{i=1}^{N_r} |f_p(t_i^r, x_i^r)|^2 \tag{10}$$

2. Mean Squared Error over Initial Data $X^o = \{(t_i^o, x_i^o)$ for i = 1 to $N_o\}$

$$L_p^o(X^o) = 1/N_o * \sum_{i=1}^{N_o} |u_p(t_i^o, x_i^o) - u_o(x_i^o)|^2 \tag{11}$$

where u_o is a known function for generating labels for initial data

3. Mean Squared Error over Boundary Data $X^b = \{(t_i^b, x_i^b)$ for i = 1 to $N_b\}$

$$L_p^b(X^b) = 1/N_b * \sum_{i=1}^{N_b} |u_p(t_i^b, x_i^b) - u_b(t_i^b, x_i^b)|^2 \tag{12}$$

where u_b is a known function for generating labels for boundary data

Each of these terms can have an associated individual weight in the final loss term.

There can be additional loss terms corresponding to other labelled data if provided.

It is this loss term which acts as a feedback mechanism from the physics-informed network that tunes the weights of the neural network and thereby ensures that the model learns the underlying physical laws while fitting to observed/known data.

The below figure comprehensively illustrates the 2-module PINN framework:

Fig 21: Building blocks of Physics-Informed Neural Networks [9]

2.5. Examples of Problems Solved

For the Schrodinger's Equation in the domain $x \in [-5, 5]$, $t \in [0,\pi/2]$

$f(t, x) = ih_t + 0.5h_{xx} + |h|^2 h$ subject to conditions (13)

$h(0, x) = 2\,\mathrm{sech}(x)$, $h(t,-5) = h(t, 5)$, $h_x(t,-5) = h_x(t, 5)$

For this problem solved in [10], the training set consisted of N_0 = 50 data points on h(0, x), N_b = 50 collocation points for enforcing the periodic boundaries and N_r = 20,000 collocation points used to enforce Eq. 13 inside the solution domain. All of them were randomly sampled using a space-filling Latin Hypercube Sampling strategy. A 5-layer deep neural network with 100 neurons per layer and a hyperbolic tangent activation function was used as the architecture to model PDE solution h (t, x). Loss function was optimised using the L-BFGS technique.

Fig 22: Top: Predicted Solution |h (t, x)| along with initial and boundary training data
Bottom: Comparison of predicted and exact solutions for three temporal snapshots [10]

[11] demonstrated usage of PINN for solving a non-linear diffusivity equation:

$$f(x, t) = \phi c_t \frac{\partial p}{\partial t} - \kappa_o \frac{\partial}{\partial x} p^2 \left(\frac{\partial}{\partial x} p\right) - g \quad (14)$$

where ϕ = initial porosity, c_t = total compressibility, p = fluid pressure, κ_o = hydraulic conductivity (scalar), g represents sink/source, $\Theta = (\phi, c_t, \kappa_o)$ is PDE parameter set

The system with domain Ω = [0, 1] and T = [0, 1] was subject to simple conditions:

IC: $p = p_0 \ in \ \Omega \ at \ t = 0$ and BC: $p = p_D \ on \ \partial\Omega_p \times T$

The number of hidden layers (N_{hl}) and the number of neurons in each layer (N_n) were taken as hyperparameters varied over a range of values with fixed number of training (N_b = 96) and collocation (N_Π = 160) points yielding least error for N_{hl} = 6 and N_n = 5. Points were sampled using Latin Hypercube and loss was minimised using L-BFGS.

Fig 23: Variation of MSE loss (split into training & collocation) with iterations [11]

PINNs have also been used to solve a system of coupled equations such as the Stokes-Darcy equation as in [18]. All loss terms corresponding to both equations are simply added to make up the final loss function using a single PINN. However, if the parameters of the PDEs are too small or if there is a discontinuity at the interface, then the PINNs don't work well for solving. Hence, the authors suggest some strategies to tackle such issues. These include adding weights to the different loss terms. This ensures that just because of the magnitude of the loss values, only certain specific terms aren't focussed upon for training. Another strategy is to use parallel network architecture where each network caters to a different sub-region of the solution. This helps improve interface discontinuity-related issues. Finally, it was suggested to have a layer-wise locally defined activation function so that the layers have different activations to capture the different aspects of complex problems.

In [14], PINN is used for the simulation of the thermochemical evolution of a composite tool material undergoing curing in an autoclave. They design a PINN with two disconnected sub-networks to model the different behaviours of the two differential equations in the coupled system: thermal conduction (heat transfer) and resin cure kinetics. Their PINN design is illustrated in the below figure.

Fig 24: Two separate networks for determining the degree of cure (α) and temperature (T). The PINN is trained sequentially, first on α and then on T, until convergence. [14]

An instance of using PINN for defect detection was found in [15] where it was employed to identify defects using pulsed thermography images of CFRP specimens. A second neural network was used as a surrogate model for determining the second-order thermal gradients (present in the PDE) for subsurface temperatures which weren't measured during experiments. The PINN reconstructs the thermogram which after background elimination reveals the anomalies present.

Fig 25: Predicted images (right 2) with the three defects as in original images (left 2) [15]

2.6. Summary & Objectives

Advances in computer vision and deep learning have enabled the extraction of useful visual features from image data and the building of highly-accurate models to perform tasks such as segmentation. Convolutional Neural Network (CNN) based models have been employed for defect detection while Recurrent Neural Networks (RNNs) are useful for sequence prediction. On the other hand, advances in the field of scientific machine learning have given rise to Physics-Informed Neural Networks (PINNs). PINNs provide a starting framework to solve ordinary and partial differential equations given initial and boundary conditions and other labelled data while being in the realm of deep learning. Examples of problems solved in literature have been provided. Accordingly, PINN will be used to solve PDEs of gradually increasing complexities, beginning with how a simple ODE can be solved using a neural network. The objective of the research is to perform experiments to identify how PINNs work and how they can be interpreted, make improvements, and demonstrate their application to model physical phenomena such as the Lorenz system, diffusion and growth. There are several design and implementation aspects concerning the PINN framework which will be investigated further herein. Hyperparameters and their possible values, tuned model and its correlation with the complexity of equation will be attempted to be determined. Other frameworks can be explored with the broader aim of generalisation (solving a family of PDEs) while also comparing them with analytical and numerical solutions. Thereafter, the best framework would be investigated and applied in-conjunction with CNN and RNN to build a data-driven (observed numerical and image data) and physics-informed (incorporated process information and physical constraints) model for defect prediction from image sequences. Once established on the microscopic scale, the idea would be extended further and applied to build an integrative model working across different (length) scales to cater to different types of defects and predict them as the material evolves.

3. Implementation & Results

Various experiments were conducted in Tensorflow on Google Colab to solve differential equations using neural networks.

3.1. Solving an ODE [24]

Let there be a simple ODE in $x \in [0, 1]$:

$$\frac{du}{dx} = f(x) = 2x \tag{15}$$

Initial Condition (IC): $u(0) = 1$

Define a trial solution g similar to Equation 6:

$$g(x) = u(0) + xNN(x) \tag{16}$$

By construction, g satisfies the IC

Objective: Train $g' = \frac{dg}{dx}$ to be equal to $\frac{du}{dx}$ in order for g to satisfy the ODE as well

First, design a neural network to model u given x

Neural Network Architecture

- Input Layer = One Neuron for x
- 2 Hidden Layers each with
  - Number of Neurons = 32
  - Activation Function = Sigmoid
- Output Layer = One Neuron for u

Randomly draw 10 collocation points from a uniform distribution

Pseudocode for Training

Repeat (for every iteration):

- For each data sample x
  - Approximate u using NN
  - Calculate the gradient of g
  - Calculate Error = g'(x) - f(x)
- Calculate Loss = MSE over samples
- Use SGD to minimise loss
- Update NN model weights

Stochastic Gradient Descent with learning rate = 0.01 was used as the optimizer

Results

Below is the loss curve obtained during the training process over multiple epochs.
The loss rapidly drops to a low value suggesting early learning with ease.
The final loss recorded at the end of 1000 epochs (training halted) was 0.0022.

Fig 26: Loss vs Epochs Curve

Equation 15 has a known analytical solution $u(x) = x^2 + 1$

100 points (x) are sampled and the g(x) values calculated from outputs of the neural network with optimised weights are plotted along with the true values u(x).

As can be seen from Fig. 27, the neural network is able to approximate u fairly well.

Fig 27: u vs x plot comparing neural network approximation with the original function

3.2. Solving a PDE using PINN

The above simple ODE was solved using the method described in [8]. This section attempts to replicate solving the Burgers' Equation using PINN as described in [12].

The Burgers' Equation is a Partial Differential Equation (PDE) commonly encountered in various areas of applied mathematics such as traffic flow, fluid mechanics, gas dynamics, and non-linear acoustics. It is given by:

$$f(t, x) = u_t + uu_x - vu_{xx} = 0 \tag{17}$$

where u (t, x) = speed of fluid, u_t represents transiency, u_x represents convection and u_{xx} represents the diffusion term with coefficient v (= viscosity taken as $0.01/\pi$)

It is valid in the domain $x \in [-1, 1]$ and $t \in (0, 1]$ subject to the conditions:

Initial Condition: $u_0 = u(0, x) = -sin(\pi x)$ (18)

Boundary Condition: $u_b = u(t, -1) = u(t, 1) = 0$ (19)

Design a neural network in tf.keras to model u (t, x) given x and t

Neural Network Architecture

- Input Layer = Two Neurons for x and t
- 8 Hidden Layers each with
  - Number of Neurons = 20
  - Activation Function = Tanh
  - Weight Initialisation Scheme = Glorot Normal
- Output Layer = One Neuron for u

50 initial, 50 boundary, and 10000 collocation points were uniformly sampled

Pseudocode for Training

Repeat (for every iteration):

- For each data sample x, approximate u using PINN
- Calculate Loss = MSE over initial & boundary data + PDE residual
- Use Adam to minimise loss
- Update NN model weights

The learning rate was decaying with iteration n in a piecewise constant fashion:
δ (n) = 0.01*1{if n < 1500} + 0.001*1{if 1500 ≤ n < 3500} + 0.0005*1{if 3500 ≤ n}

Results

The training was performed for 5000 epochs using both SGD and Adam with the final loss 0.00051 for Adam being much lesser than 0.1165 for SGD. The loss gradually falls with epochs after an early steep fall. It undergoes a few fluctuations after which it keeps on decreasing indicating learning still continues albeit slowly.

Fig 28: Loss vs Epochs Curve for PINN trained using Adam

The trained PINN was used to predict u over the domain grid of x and t points:

Fig 29: BE solution u (t, x) predicted by PINN plotted over the domain grid. Shock formation begins at t = 0.4s and ends up in a steep gradient at x = 0 which is similar to as noted in [12].

Changing the Conditions

Keeping the PDE same as earlier, the PINN was re-trained for modified conditions:

Initial Condition: $u_0 = u(0, x) = sign(x)$ (20)

Boundary Condition: $u_b = u(t, \pm 1) = \pm 1$ (21)

The PDE solution should be a slackening step function for these conditions

The loss curve obtained was similar as earlier with the final loss being 0.00025

Fig 30: 3D view of PINN predicted solution u (t, x) plotted over the domain grid.

Fig 31: Time-variation of u as a function of x shown as a 2D plot.
As expected, the gradient of the step at x = 0 keeps on decreasing.

Modifications

Uniformly sampled validation data was added in a 1:5 ratio wrt the earlier data.

L2 weight regulariser = 0.01 and dropout = 0.2 were added to each NN layer.

The PINN was trained as earlier using Adam for 5000 epochs but now the loss over unseen validation data was also recorded after training for each epoch.

Fig 32: Training and Validation Loss Curves vs Epochs

The validation loss closely follows the training loss indicating there is no overfitting. The final training loss = 0.000657 and the final validation loss = 0.000689.

3.3. Predicting Evolution of Solution

In [7], images were generated using PDE-based simulations and microstructure evolution was independently predicted using RNN on convoluted image features without the network explicitly having the knowledge of the PDE. Similar to that case, if the physics-informed neural network is used to model the solution u and the prediction dataset is saved, then, in the absence of the PINN or the PDE being obeyed by PINN, it is desirable to predict the evolution of u with time.

The PINN predicted u values over the domain grid points (500 x 500) were saved in a tabular format. The dataset was split into train ($0 < x < 0.5$) and test ($0.5 < x < 1$) sets of equal size = 250. The value of u over the initial half (250) timesteps (till t = 0.5s) was taken as input data while the value of u over the other half (250) timesteps (above t = 0.5s) was taken as the output label data. An LSTM network was used for forecasting u (x, t). This is illustrated in Fig 33.

Fig 33: Illustration of the dataset and how it can be used for forecasting using LSTM

Neural Network Architecture

- LSTM Layer = 32 LSTM units
- Dense Layer = 250 neurons

The network was trained for 50 epochs with batch size = 10 and validation split = 0.3

Pseudocode for Training

Repeat (for every iteration):

- For each data sample x in the batch
  - Use half of PINN u(x) time-series as input
  - Predict other half time-series using LSTM
- Calculate Losses = MSE between LSTM predictions and PINN labels
- Use Adam to minimise training loss
- Update LSTM model weights

Results

The training and validation loss curves in Fig 34 are aligned although the training loss (final MSE = 0.0011) majorly stays below the validation loss (final MSE = 0.007). The model was evaluated on the test dataset yielding MSE = 0.0019. This indicates the absence of overfitting.

Fig 34: Training and Validation Loss Curves vs Epochs

3.4. Linear Convection Diffusion PDE

If the Burgers' Equation (Eq. 17) is linearised by making the coefficient of u_x to be a constant c, then the PDE becomes a linear convection-diffusion equation:

$$f(t,\ x)\ =\ u_t + cu_x\ -\ vu_{xx}\ =\ 0 \tag{22}$$

The analytical solution for the simple case c = v = 1 and conditions Eq. 18 & 19 is:

$$u\ (t,\ x)\ =\ \ -\ e^{-\pi^2 t} sin(\pi x) \tag{23}$$

The PINN for solving Eq. 22 was trained similar to the case for solving Eq. 17

Fig 35: 3D view comparing analytical and PINN solution

The solution predicted by PINN closely overlaps with the analytical solution

The final loss after training of PINN = 0.00014. The Mean Absolute Error (MAE) between the PINN predicted and analytical u values is MAE = 0.0015.

Fig 36: Temporal snapshot of the 2 solutions u(x) at t = 0.2s

Error between the predicted and actual values of u is clearly visible in Fig 36 mainly in the middle values of x. This is expected as the collocation (domain) data isn't used for training. But the nature of the solution curve is well-predicted as per the PDE.

Fig 37: Variation of u (predicted and analytical) with x every 0.1s.
Note that the solution curve flattens out with time.

In a fresh training, the three individual loss terms were also recorded. The final training loss = 10.99E-5 was composed of initial data MSE loss = 2.227E-5, boundary data MSE loss = 2.737E-5, and PDE residual loss = 6.027E-5. While, the final validation loss = 11.34E-5 was composed of initial data MSE loss = 3.515E-5, boundary data MSE loss = 1.764E-5, and PDE residual loss = 6.062E-5.

3.5. PINN Design & Implementation

In order to come up with the most appropriate PINN architecture that fits the data and follows the PDE, several values are possible for various modifiable parameters (called hyperparameters) which affect the training process. There could also be minor modifications in the approach or methodology for different aspects associated with PINNs. Varying any of them will result in a different PINN solution for a given problem. These include data sampling strategy, neural network architecture details, loss function design, and optimisation strategy.

3.5.1. Data Distribution and Sampling

In this study, wherever required, data has been sampled randomly from a uniform distribution for simplicity although Latin Hypercube Sampling is popular in literature. The training data associated with the conditions are generally few scattered observations (here 50 initial and 50 boundary points) while a greater number of collocation points (here 10000) are chosen from the domain to evaluate the residual.

3.5.2. Loss Function Design

The general loss function defined in Fig 21 is for the case of enforcing soft-boundary constraints, meaning the neural network learns to fit the training data for the conditions gradually but it isn't guaranteed that all conditions will be exactly satisfied. Further, the weights associated with the different loss functions are hyperparameters which affect the learning efficiency, and determining them is a complexity to be addressed. For flexibility though, the weight associated with the PDE residual will help to vary the fidelity of the PDE model. Specific components of the neural network can be made to purely meet the (exact or hard) conditions a priori such as by using distance functions as in [16]. Here, the loss function as defined in Eq. 9 is considered which is calculated using automatic differentiation of neural network outputs.

3.5.3. Optimisation

The PINN loss function in most literature is optimised via minibatch sampling using Adam or the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm which will yield one of several local minima. Here, the optimizer is chosen to be Adam as a straightforward implementation for it is available in the TensorFlow library. Adam, which combines adaptive learning rate and momentum methods is known to have higher convergence speed compared to SGD and had also resulted in lower loss while solving the Burgers' Equation. A decreasing learning rate is popular for faster convergence away from minima while the stopping criteria could be a fixed number of iterations or a lower bound on the loss. However, such parameters will have similar effects on the model as it varies. Hence, in order to compare different models, these will be kept fixed to be the same as earlier. The neural network model parameters thus are the most significant hyperparameters, which need to be optimised before other aspects can be explored to get the most accurate solution.

3.5.4. Neural Network Architecture

Most of the PINNs in literature are single feed-forward neural networks. There have been instances of wide networks (say 100 neurons per layer as in [10]) as well as deep networks (say >5 layers). Generally, more hidden layers are required to model complicated non-linear relationships but this results in high training costs and efficiency issues (in terms of computing time and memory). Hence, shallow networks (say <5 layers) have also been used. In fact, sparse architectures designed using kernel networks and yielding interpretable results (SPINNs) were proposed in [13] which act as a bridge between traditional meshless methods and deep neural networks for solving PDEs. There also have been instances of the use of multiple fully-connected networks mostly to approximate individual equations which are blended together to solve a larger mathematical model. These include [14] & [15].

Here, a single fully-connected feed-forward neural network is chosen with the number of hidden layers (h) and the number of neurons in each layer (n) taken as hyperparameters for tuning. h is allowed to be any integer from 1 to 20 while n takes up a multiple of 4 (for faster convergence via matrix multiplications by the computing hardware) in the range [4, 256]. For combinations of h and n, which result in big networks with many trainable parameters, there is a possibility of overfitting. To compensate, L2 weight regularizer with penalty value among {0, 10E-1, 10E-2, 10E-3, 10E-4, 10E-5, 10E-6} and dropout fraction among {0, 0.1, 0.2, 0.3, 0.4, 0.5} is considered for each layer and taken as a hyperparameter.

Activation functions are the nonlinear transformations introduced in every layer of the neural network and functions such as ReLU, Sigmoid, Tanh, and Swish are found in the PINN literature with the infinitely differentiable Tanh being the most popular. Here, the activation function (a), same for every layer, is taken as a hyperparameter. The weight initialisation scheme is crucial to determine the convergence rate of training as along with the activation, it plays a major role in any vanishing or exploding gradient issues with the network. ReLU is considered to be robust to such issues and earlier studies have shown He weight initialiser works best for it. For other activation functions, Glorot weight initialiser is assumed and the type of distribution (Normal or Uniform) from which the initial weights are sampled is taken as a hyperparameter.

3.5.5. Error Analysis

The accuracy of a PINN can be measured in terms of the error which can be decomposed into three terms: approximation error, optimisation error, and the generalisation error. The approximation error is the error associated with the neural network approximation and this can be measured wrt the available exact values. As the loss function is non-convex, optimisation will not always yield the global minimum. The error associated with this is the optimisation error which can't be determined. The error measured in the predictions by a trained network wrt unseen

data is the generalisation error. [9] Here, the MSE loss on labelled data recorded during training is the approximation error while the MAE evaluated on test data such as the domain grid is an estimate of the generalisation error. Bounds need to be established on these errors to design PINNs with sufficient accuracies.

3.5.6. Generalisability

The generalisation aspect in terms of the capability of a single model to solve a variety of equations, say a family of PDEs needs to be investigated. In this regard, in [19], researchers proposed the DeepONet framework which is based on the operator approximation properties of neural networks. It consists of two sub-networks: one called branch-net for encoding the input functions and another called trunk-net for encoding the locations of the output functions. Once trained on inputs and outputs for a set of PDEs, it will have learnt the operator for that family of PDEs. So, if it is provided with data representing a new PDE (belonging to that family), it yields data representing the solution to that PDE. [22] This operator learning framework has been demonstrated for predicting multiscale bubble growth dynamics in [20].

3.5.7. Tuning using Bayesian Optimisation

Heuristics as mentioned earlier were used to reduce the high-dimensional search space for hyperparameter optimisation which is a complexity with slow training itself being an issue in PINNs. While grid search becomes computationally impractical for searching among a range of continuous values, random search could be a possible option. However, a more subtle approach that takes into account results after every iteration to optimise the search process is desirable. The gaussian-based bayesian optimization (BO) is one such technique. It is a sequential design strategy for the global optimization of black-box functions. [23] It works by constructing a posterior distribution for the associated function, which is improved as the number of observations grows with iterations resulting in a more restricted parameter search space each time. It also provides flexibility to the user to decide how to balance random exploration with the exploitation of learnings from the results of prior iterations. It has been used for hyperparameter tuning of PINNs such as in [17] for solving Helmholtz problems. They used 5 hyperparameters: learning rate for optimisation [10E-4, 5E-2], number of NN layers [1-10], number of neurons in each layer [5,500], activation function {sin, sigmoid, tanh}, and the weight associated with the boundary loss term [1, 10^7]. From the results, they concluded that wide shallow networks with sin activation perform better at solving the Helmholtz problems.

Here, BO is used with the PINN for Eq. 22 as the black-box function and the negative of the training loss as the objective to be maximized. 10 iterations of BO were performed with an additional 5 for random exploration. The following table lists the NN model hyperparameters finally considered and the values they can take up.

Hyperparameter	Symbol	Range/Set of Values
No of Hidden Layers	h	Integer in [1, 20]
No of Neurons	n	Multiplier of 4 in [4, 256]
Dropout Fraction	d	{0, 0.1, 0.2, 0.3, 0.4, 0.5}
Activation Function	a	{relu, sigmoid, tanh, swish}
Weight Initialiser	wi	If ReLU then He else Glorot {normal, uniform}
L2 Weight Penalty	wr	{0, 10E-1, 10E-2, 10E-3, 10E-4, 10E-5, 10E-6}

Table 1: Search space for Hyperparameter Tuning

The least training loss for optimal combination of hyperparameters = 3.354E-5

3.5.8. Interpretation

In order to improve the interpretability of PINN so as to determine how to design the neural network to solve any given PDE, an experiment was carried out. The terms in the family of PDEs consisting of equations such as Eq. 17 and Eq. 22 were varied and their effect on the hyperparameter values of the tuned network was noted for the case of the unsteady state. The results are tabulated in Table 2.

The following observations are noted from this experiment:

1. The optimal number of hidden layers increases to capture complexities such as presence of more terms, non-linearity, double-differentials in the PDE.
2. Using Swish activation results in a higher performance as compared to others.
3. For deeper networks, dropout gets enabled possibly to avoid overfitting.
4. Regularisation is applied only if the model has a very high number of weights.

Equation	Order	No of Hidden Layers	Activation	Dropout	Weight Initialiser	Weight Penalty
$u_t + u_x - u_{xx} = 0$	2	7	Swish	0.5	Glorot Normal	0
$u_t - u_{xx} = 0$	2	5	Swish	0	Glorot Normal	1.00E-06
$u_t = 0$	1	4	Swish	0	Glorot Uniform	0
$u_t + uu_x - u_{xx} = 0$	2	14	Swish	0.5	Glorot Normal	0
$u_t + uu_x = 0$	1	7	Swish	0.5	Glorot Normal	0

Table 2: Tuned Hyperparameter Results for various PDEs

4. Summary & Conclusion

In order to build an intelligent defect prediction model, this work has investigated literature with a focus on Physics-Informed Neural Networks. PINNs have been used to build models that obey physical laws (differential equations and known conditions) within the data-driven framework of deep learning. The need for such a framework was established in comparison with previous applications of ML in microstructural studies, especially CNN-based defect detection models. Additionally, predicting the evolution of microstructure using RNN with respect to image sequences and PINN solutions was discussed. This indicates the possibility of a combined CNN-PINN-RNN framework to meet the project's end goal: CNN to identify features from image data, PINN to infuse physics and known information and learn from (numerical) labelled data, and RNN to predict evolution (extrapolation) with time.

This report has discussed PINNs in detail: understanding of the framework, design aspects, and implementation aspects. After looking at problems solved previously in the literature, experiments were performed to solve ODEs and PDEs. Results on validation data (with losses of order 10E-4) suggest the strong generalisation performance of PINN, while also revealing inaccuracies mostly for interior collocation data points. A glimpse of solving coupled equations, complex systems and family of PDEs was provided which will be delved into in future. After a discussion on hyperparameters, tuning was performed using bayesian optimization, which in just 10 iterations reduced the losses by an order. An experiment to interpret the modelling of PDEs by PINN was performed which will be extended further to several equations in order to build the base for the design of a PINN given any equation.

4.1. Advantages of PINNs [21]

- Highly fast and accurate prediction: Once a PINN is trained, the prediction is almost instantaneous and can be highly accurate, thanks to the learning and approximation properties of neural networks. In contrast, numerical solutions are very time-consuming.
- Highly stable and efficient solving of even complex problems: PINNs don’t suffer from stability issues or time-step limitations of numerical approaches. Some PDEs are notoriously difficult to solve using classical methods but PINNs provide an efficient framework that can be used to solve any PDE. The power of deep learning helps tackle the curse of dimensionality, so PINNs can be used to solve even high-dimensional complex problems.
- Work well with incomplete models and imperfect data: It is possible to build models which yield meaningful solutions even when some parameters in PDEs are unknown or conditions are absent - realistic scenarios where classical approaches fail. This is enabled by smoothness/regularity in the loss function.

- Strong generalisation in small data regime: By embedding physics, deep learning models are effectively restricted to lower dimension solution search space and thus can be trained with smaller data. This means, PINNs are more accurate with low generalisation errors as compared to purely data-driven counterparts.
- Helpful in understanding deep learning: The PINN framework helps to 'unbox' the deep learning models at play with the infusion of physical laws. This enables better interpretability of the model built for the system and helps reveal insights into the inner mechanisms and effectiveness of deep learning workflow.

4.2. Challenges

Although PINNs find several applications and can be used to solve any differential equation, the design of a PINN itself differs with each physical problem and requires a pretty nuanced understanding of its working at the fundamental level. It also needs to be retrained for any modifications in the underlying rules or conditions. The most important challenge for PINNs is to establish their convergence to the correct solution of the differential equation. First, the approximation error has to tend to zero because the labels and conditions are generally well-defined. The slow training and tuning of PINN also needs to be addressed. A good optimization procedure with training for larger epochs can be helpful to reduce the error and converge faster. Thereafter, the focus should be on achieving a low generalisation error which was present as seen in Fig 36 for the interior domain points. Tackling overflow issues during bayesian optimization is also a challenge. Finding the right values for different hyperparameters in the complete approach (such as weights in the loss function) and establishing solutions to a defined tolerance is still under research. Interpretation of a PINN, the generalisation of its framework to solve a family of PDEs, and improvements in its implementation aspects are desirable research topics.

4.3. Future Steps

The following steps are conceived to be executed in the near future as part of this research project:

- Applying PINN to solve coupled equations and complex systems using strategies outlined in the Literature Review section.
- Extending the interpretation experiment to several equations starting from basic linear & quadratic equations, moving onto higher degree polynomial equations, then simple ODEs & PDEs, moving onto higher orders, and finally complex systems to generate enough data points for meaningful conclusions.
- Exploring frameworks to solve any general equation or a family of PDEs.
- Quantifying the benefits of PINN over numerical approaches wrt the cost of training and cost of running the prediction via multiple experiments.

5. References

1. M Shen, G Li et al., “Multi defect detection and analysis of electron microscopy images with deep learning”, Computational Materials Science, Vol. 199 (2021)

2. G Roberts, SY Haile, R Sainju et al., “Deep Learning for Semantic Segmentation of Defects in Advanced STEM Images of Steels”, *Sci Rep*, Vol. 9:12744 (2019)

3. F Nikolić, I Štajduhar, M Čanađija, “Casting Defects Detection in Aluminum Alloys Using Deep Learning: a Classification Approach”, Inter Metalcast (2022)

4. Q Boyadjian, N Vanderesse, M Toews, P Bocher, “Detecting Defects in Materials Using Deep Convolutional Neural Networks”, Lecture Notes in Computer Science, Springer, Vol. 12131 (2020)

5. A Chowdhury, E Kautz, B Yener, D Lewis, “Image driven machine learning methods for microstructure recognition”, Computational Materials Science, Vol. 123 (2016)

6. H Chan, M Cherukara, TD Loeffler et al., “Machine learning enabled autonomous microstructural characterization in 3D samples”, npj Computational Materials, Vol. 6.1 (2020)

7. K Yang, Y Cao, Y Zhang et al., “Self-supervised learning and prediction of microstructure evolution with convolutional recurrent neural networks”, Patterns, Vol. 2.5 (2021)

8. IE Lagaris, A Likas, DI Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE Transactions on Neural Networks, Vol. 9.5 (1998)

9. S Cuomo, VS Di Cola, F Giampaolo et al., “Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What’s Next”, Journal of Scientific Computing, Vol. 92: 88 (2022)

10. M Raissi, P Perdikaris, GE Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”, Journal of Computational Physics, Vol. 378 (2019)

11. T Kadeethum, TM Jørgensen, HM Nick, “Physics-informed neural networks for solving nonlinear diffusivity and Biot’s equations”, PLoS ONE, Vol. 15.5 (2020)

12. J Blechschmidt, OG Ernst OG, “Three ways to solve partial differential equations with neural networks — A review”, GAMM-Mitteilungen (2021)

13. AA Ramabathiran, P Ramachandran, “SPINN: Sparse, Physics-based, and partially Interpretable Neural Networks for PDEs”, Journal of Computational Physics, Vol. 445 (2021)

14. SA Niaki, E Haghighat et al., “Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture”, Computer Methods in Applied Mechanics and Engineering, Vol. 384 (2021)

15. WH Lim, S Sfarra, Y Yao, “A Physics-Informed Neural Network Method for Defect Identification in Polymer Composites Based on Pulsed Thermography”, Engineering Proceedings, Vol. 8:14 (2021)

16. N Sukumar, A Srivastava, “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”, Computer Methods in Applied Mechanics and Engineering, Vol. 389 (2022)

17. Paul Escapil-Inchauspé, Gonzalo A. Ruz, “Hyper-parameter tuning of physics-informed neural networks: Application to Helmholtz problems”, arXiv:2205.06704v1 (2022)

18. R Pu, X Feng, “Physics-Informed Neural Networks for Solving Coupled Stokes–Darcy Equation”, Entropy, Vol. 24:1106 (2022)

19. L Lu, P Jin, G Pang et al., “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators”, Nature Machine Intelligence, Vol. 3 (2021)

20. C Lin, Z Li, L Lu et al., “Operator learning for predicting multiscale bubble growth dynamics”, The Journal of Chemical Physics, pp. 154, 104118 (2021)

21. GE Karniadakis, IG Kevrekidis, L Lu, et al., “Physics-informed machine learning”, Nature Review Physics, Vol. 3, pp. 422–440 (2021)

22. A Ananthaswamy, “Latest Neural Nets Solve World’s Hardest Equations Faster Than Ever Before”, Quantamagazine (2021) https://www.quantamagazine.org/latest-neural-nets-solve-worlds-hardest-equations-faster-than-ever-before-20210419/

23. F Nogueira, “Bayesian Optimization: Open source constrained global optimization tool for Python”, Github (2014) https://github.com/fmfn/BayesianOptimization

24. C Davi, “Using Neural Networks to solve Ordinary Differential Equations”, Towards Data Science (2021) https://towardsdatascience.com/using-neural-networks-to-solve-ordinary-differential-equations-a7806de99cdd