Eötvös Loránd University

Faculty of Science

Institute of Mathematics

Neural network solution of the inverse heat conduction problem

Supervisor:
Dr. Ferenc Izsák
Habil. Associate Professor

Author:
Patrik Roland Czakó
Postgraduate student

Budapest, 2023

Abstract

This thesis presents a novel approach to address the Inverse Heat Conduction Problem (IHCP) by designing a neural network architecture capable of solving the problem on different scales. Various network architectures were explored using the analogy of finding inclusions in cast metal based on temperature measurements. The results showed that convolutional architectures outperformed Multilayer Perceptrons in solving the IHCP, achieving accurate predictions on coarser validation datasets. However, poor generalization to higher-resolution data was observed due to limited training data variability and the risk of overfitting. To overcome this challenge, a multigrid-inspired approach was proposed, incorporating an autoencoder-based network. The autoencoder learned a compressed representation of the input data, symbolizing the coarser scale, and this representation was then used in a pretrained convolutional network for improved performance on finer data. The proposed architecture demonstrated enhanced performance on higher-resolution data, leveraging the multigrid-inspired restriction operator provided by the autoencoder. This research contributes to the field of deep learning by addressing the challenges of transferring knowledge across scales and improving the generalization of neural networks.

Contents

Chapter 1

Introduction

During the casting of metals, air bubbles are often formed, which are called inclusions. This phenomenon occurs when gases, such as oxygen or hydrogen, dissolve in the liquid metal and are released within the metal's interior during the cooling process. Although there are several methods to detect inclusions in metals, such as X-ray radiography or ultrasonic testing, we attempt to approach this problem by inverse heat conduction and design a neural network that can accurately detect inclusions.

1.1 The inverse heat conduction problem

The inverse heat conduction problem (IHCP) involves determining the internal heat transfer characteristics of a material based on measurements of its surface temperature. This problem has important applications in materials science, thermal engineering, and other fields. There are several papers that have worked on inverse heat conduction problems (see, for instance, [1, 2, 3, 4, 5]).

Direct heat conduction problem is considered to be mathematically *"well-posed"*, i.e., error in the input data (such as boundary or initial conditions) of the heat conduction model will induce an error in the same order of magnitude in the calculated temperature distribution. However, the majority of the inverse problems, including IHCP are considered to be mathematically *"ill-posed"*. In these types of problems, knowledge is lacking in either boundary or initial conditions, or thermal properties of the body. Therefore, these unknowns need to be estimated using measured temperature data within the body of the domain. Being mathematically ill-posed means

that unavoidable random errors (noise) in the measured data may induce errors amplified by several orders of magnitudes in the estimated unknowns. It is mainly for this reason that IHCP is considered to be considerably more *"challenging"* than the direct heat conduction problem [6].

A formal mathematical model of an inverse problem can be derived with relative ease. However, in most cases, the process of solving the inverse problem is extremely difficult and the so-called exact solution practically does not exist. In this way, for the numerical solution of inverse problems, a family of approximation methods were employed: iterative procedures, regularization techniques, stochastic and system identification methods, methods based on searching an approximate solution in a subspace of the space of solutions (if the one is known) or a combination of the above techniques [7].

In particular, if the heat map is known only in certain points of the domain Ω, two different heat conduction parameters may result the same temperatures in the given points. Also, in general, the continuous dependence on the data is failed. This means that in case "nearby" or very similar observations, we can have quite different heat conduction parameters. As we intend to work with approximations, both of these problems could harm any solution algorithm. Therefore, in the conventional numerical methods, a so-called regularization procedure is carried out. The paper [8] introduces the concept of optimization methods as a powerful tool for solving ill-posed problems. These methods involve minimizing a functional that consists of a data fidelity term and a regularization term. The regularization term incorporates prior knowledge about the solution ensuring a unique minimum and can be chosen differently based on the problem at hand.

1.2 Neural networks for inverse problems: an overview

In recent years, neural networks have emerged as a powerful tool for solving inverse problems in various fields. In the context of the inverse heat conduction problem, neural networks have been used to predict internal temperature distributions based on surface temperature measurements. The ability of neural networks to learn complex patterns and relationships in data makes them well-suited for this

task.

This method appeared already in 1999, when Krejsa et al. introduced two approaches for using neural network to solve the inverse heat conduction problem [9]. The first is called Whole History Mapping (WHM) approach, which involves mapping the whole vector of observed temperature history (covering the time span of measurements) to the corresponding vector of heat transfer coefficients at the corresponding times. Whereas, the second one is the Sequential Approach, where the local time changes in sensor temperature are mapped to the corresponding local time value of surface heat flux. This article presents the relative advantages and limitations of both approaches. Furthermore, the following approach-independent steps for solving the IHCP using neural networks are outlined:

1. *Training set generation.* Training sets (pairs of input and output vectors) are generated by any forward solution method for the case considered. According to the paper, the heat transfer coefficients should be assumed either generated randomly or by using *a priori* knowledge about the target process. Whereas a superposition method (such as Duhamel's theorem) or a numerical procedure (such as Finite Element Method). In this way, one can recognize the collection of temperature recordings at the sensor location as the training input vectors, and the corresponding history of the assumed heat transfer coefficients as the training output vectors.

2. *Building the network.* The design of the network considering the used approach, learning algorithm (Back-propagation, Radial Basis Functions, Cascade Correlation, Genetic Algorithm and Adaptive Linear methods were used in the paper) and the size of the network (depth and width).

3. *Training the network.* In the training phase, the network tries to "learn" facts from the training set. Learning is controlled by the learning algorithm. Training is stopped when the error is reasonably small or it is no longer decreasing.

4. *Testing the network.* In testing phase, input vectors not contained in the training set are used at the input layer and the corresponding heat transfer coefficients or heat flux result is are used in the output layer. In a real problem, the temperature time history would be determined from an experiment. Usually,

a few examples for which the output is known are used to test the network's ability to generalize learned facts.

Although, since then, the technology of neural networks undergone significant improvement, these steps still apply. Modern applications also follow this scheme, as I will do in my work. In this article, data were generated by simulating the heating of a chrome-steel slab insulated on one side, while temperature values were measured close to the heated surface. The heat transfer coefficient was given a specified time variation that they aimed to predict using neural networks. Both of the methods showed some promise for the solution of inverse problems. However, they also note that further research is needed to fully explore the potential of neural networks in solving the IHCP.

So far, many authors have approached the problem in different ways using different network architectures [10, 11, 12, 13, 14]. Cortés-Aburto et al. achieved promising results at recovering the internal heat generation source in cylindrical coordinates with a multilayer perceptron feed-forward neural network trained via back-propagation algorithm [10]. Mikki et al. also achieved good results using similar methods on a different approach to the problem, they used feed-forward neural networks to determine the initial temperature distribution on a slab with adiabatic boundary conditions, from a transient temperature distribution, obtained at a given time [12].

In contrast to the studies presented so far, Qian et al. constructed a so-called physics-informed neural network [14]. This architecture consists of a fully-connected deep neural network and physics constraints, which was used to approximate the solution of a one-dimensional heat conduction equation and equipped with boundary and initial conditions. They also investigated the effect of different activation functions on the model's performance. Based on their results, the Hyperbolic Tangent (Tanh) activation function fits best the problem. In their physics-informed approach, the first and second derivatives are used to calculate the loss terms needed to train the neural network. Therefore, setting first or second derivatives to zero would make the training ineffective. Accordingly, they attribute the relatively poor performance when using Rectified Linear Unit (ReLU) activation functions to this phenomena. The Sigmoid activation function has non-zero first and second derivatives (except when $x = 0$). However, it suffers from the vanishing gradient problem [15] more than

the Hyperbolic Tangent.

Jardim et al. investigated the identification of contact failures in the bonding process of different materials [13], where two materials are joined with an adhesive layer. Therefore, gaps of air can form in this layer, meaning that the adhesion process is not perfect. This problem is very similar to the topic of this study. They used a neural network modeled as an inverse heat conduction problem to estimate the positioning of gaps. The input consists of transient temperature measurements that are encoded into a latent space of a denoising autoencoder [16], which is used with the purpose of reducing the dimension and noise on the experimental data. These reduced data are used as input to a fully connected multilayer perceptron network. The authors point out that this approach is promising due to the good accuracy and low computational cost observed. By including different noise levels within a defined range in the training process, the network can generalize the experimental input data and estimates the positioning of defects with similar quality.

1.3 Research objectives

The main objective of this study is to develop a neural network architecture that is able to detect inclusions in cast metal cubes based on surface temperature measurements, that is solving the inverse heat conduction problem. Specifically, we aim to:

1. Simulate directly the heat conduction in a heterogeneous metal cube which is heated from below and has insulated sides, so that obtain the temperature on its top plane.

2. Develop a neural network that can accurately detect the size and position of the inclusions based on input temperature measurements on the top face.

3. Provide a solution for detecting inclusions based on data measured on a finer grid, which is more computationally expensive to simulate directly. In this case, only coarse data is available to train the model, thus making the generalization even more difficult.

Overall, this study aims to contribute to the development of a reliable and efficient neural network based method for solving the inverse heat conduction problem using the analogy of a cast metal cube with its inclusions.

Chapter 2

Data generation

As it was mentioned in chapter 1, the input of the neural network is a couple of temperature vectors at certain observation times measured on the upper side of the cube and the network predicts the location and size of inclusions in the cube. In order to train and evaluate such a network, a dataset of temperature measurements and inclusion locations must be available. For this purpose, we simulated heat conduction with predefined inclusions of different locations and sizes in the metal cube solving the heat equation with Finite Difference Method. This chapter contains a description of the data generation method, highlighting the design decisions and their justification.

2.1 The heat equation

In order to simulate heat conduction and calculate the temperature on the upper plane of the cube under the defined conditions, we used the Fourier–Biot equation:

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} + \frac{e_{\text{gen}}}{k}, \tag{2.1}$$

where

- $\alpha(x, y, z)$ is the thermal diffusivity,
- $T(t, x, y, z)$ is the temperature,
- t is time,
- x, y, and z are the Cartesian coordinates,
- $e_{\text{gen}}(x, y, z)$ represents any internal heat generation in the element, and
- $k(x, y, z)$ is the thermal conductivity.

This is a partial differential equation that describes the behavior of temperature in the domain. It combines Fourier's law of heat conduction with the Biot's law of convective heat transfer, that is the equation (2.1) considers both the conductive heat transfer within the material and the convective heat transfer at the material's boundaries.

Since in my case there is no chemical reaction, nuclear reaction, or other processes that generate heat within the material, we assumed the absence of heat generation, that is $e_{\text{gen}} = 0$. In this way, incorporating the initial and boundary conditions, and using the notation Q for the cube, Q_{top} for the top face, Q_{s} for the side faces and Q_{b} for the bottom face, respectively, the mathematical model of the heat conduction over the time interval $(0, t_{\max})$ can be given as follows.

$$\begin{cases} \frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \quad (x,y,z) \in Q,\ t \in (0, t_{\max}) \\ T(0,x,y,z) = T_0(x,y,z) \quad (x,y,z) \in Q, \\ T(t,x,y,z) = T_{\text{top}}(t,x,y,z) \quad (x,y,z) \in Q_{\text{top}},\ t \in (0, t_{\max}), \\ T(t,x,y,z) = T_{\text{b}}(t,x,y,z) \quad (x,y,z) \in Q_{\text{b}},\ t \in (0, t_{\max}), \\ \mathbf{n} \cdot \nabla T(t,x,y,z) = 0 \quad (x,y,z) \in Q_{\text{s}},\ t \in (0, t_{\max}), \end{cases} \tag{2.2}$$

where $\mathbf{n}$ denotes the outward normal vector at the boundary. In course of the direct heat conduction problems, $T_0, T_{\text{top}}, T_{\text{b}}$ and α are given, and all of the other quantities can be determined. In the inverse problem, we investigate, T_{top} is given at certain time points $(t_1, \ldots, t_i)$, while α is to be recovered.

2.2 Numerical simulation

Simulating heat conduction in 3D involves solving the heat equation (2.2) numerically using some approximation method. There are various approaches to solve numerically the heat equation, such as finite difference method, finite element method, and boundary element method. In my work, we used a finite difference method for this purpose. This section describes the numerical simulation in details including the discretization, initial and boundary conditions and stability condition.

2.2.1 Geometry and boundary conditions

The first step of the simulation was to define the physical domain of the problem and the boundary conditions. The domain is a heterogeneous cast metal cube of

size L with inclusions, i.e. $Q = (0, L) \times (0, L) \times (0, L)$. The cube is heated from one direction at a constant temperature, while the sides of the cube are insulated. The upper plane of the cube is in contact with a room-temperature environment, where the measurements are recorded.

Since we would like to simulate heat conduction on a scale of meshes, we introduced a new variable n meaning the number of grid points in the cube. The white area in Figure 2.1 shows the mesh of the discretized cube in case of $n = 6$. δ denotes the distance between the grid points which can be calculated as $\delta = \frac{L}{n-1}$. Dirichlet boundary condition was used in the simulation process, which specifies the temperature of the object at the boundary. For this purpose, the cube was padded with an additional grid point from each direction. This new padded 3D mesh of size $n + 2$ stores the temperatures which is shown in Figure 2.1.

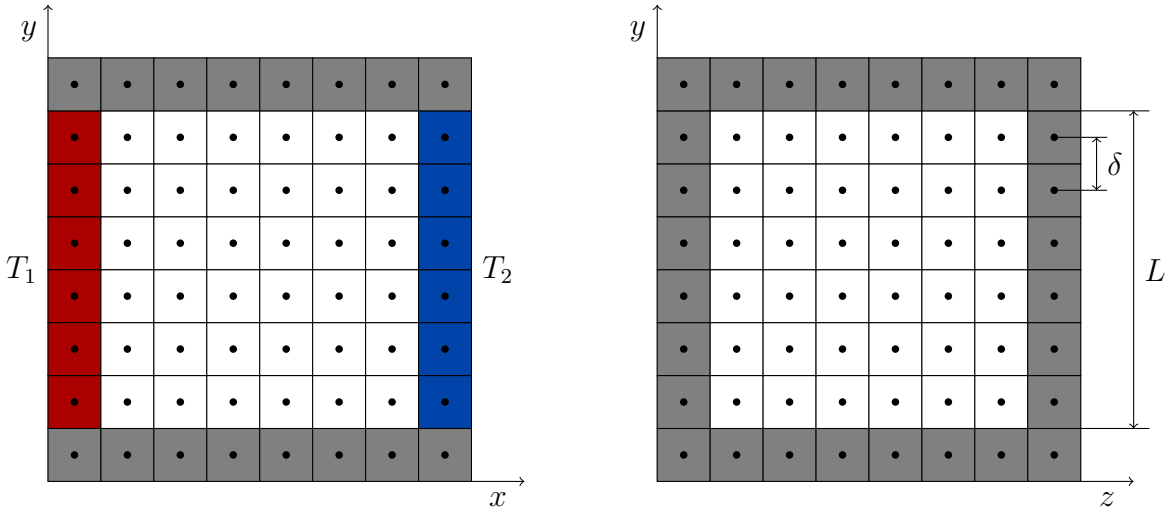

Figure 2.1: Mesh of the discretized domain ($n = 6$). **Left:** projected perpendicular to the direction of heat conduction. **Right:** projected in the direction of heat conduction. **White:** discretized metal cube with a size of L. **Red:** Heat source with a constant temperature of T_1. **Blue:** room temperature (T_2) environment. **Gray:** insulating padding. δ denotes the distance between the grid points.

According to the initial conditions, the entire mesh is initially at room temperature, except for the heat source (indicated in red), which is at temperature T_1. This temperature remains constant to represent the continuous heat flow during the simulation process. Similarly, the upper environment of the cube (indicated in blue) remains at room temperature. Finally, the insulation condition is met by adjusting the temperature of the insulating padding grid points (indicated in gray) in every iteration to be equal to their neighboring grid point in the cube. In this way, the

corresponding temperature difference on the side faces becomes zero. As a result, the heat flux is also zero at these faces.

One more parameter needs to be defined to solve the heat equation from (2.2), which is nothing but the thermal diffusivity tensor α with a shape of (n, n, n). This tensor contains the thermal diffusivity values of each grid point in the cube. The vast majority of its values have the thermal diffusivity of cast metal, while some values correspond to air representing inclusions.

2.2.2 Iteration over time

In order to solve the heat equation (2.2) and update the temperature array in each iteration, we used forward-time central difference method, which is a numerical technique used to approximate the solution of partial differential equations, such as the heat equation. It is a finite difference method that discretizes both time and space to approximate the derivatives in the equation.

The second-order spatial derivatives in the heat equation (2.2) represent the rate of change of temperature with respect to position in each direction. We approximated these derivatives using the central difference formulas in Equation (2.3), (2.4) and (2.5), which were derived using a Taylor series expansion of $T((x_i, y_j, z_k) + \Delta x)$, $T((x_i, y_j, z_k) - \Delta x)$, $T((x_i, y_j, z_k) + \Delta y)$, $T((x_i, y_j, z_k) - \Delta y)$, $T((x_i, y_j, z_k) + \Delta z)$, and $T((x_i, y_j, z_k) - \Delta z)$ around the point (x_i, y_j, z_k), which, for the brevity will be denoted with (i, j, k). Also, the time is fixed and the corresponding coordinate is omitted in the following derivations.

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T(i+1,j,k) - 2T(i,j,k) + T(i-1,j,k)}{(\Delta x)^2} \tag{2.3}$$

$$\frac{\partial^2 T}{\partial y^2} \approx \frac{T(i,j+1,k) - 2T(i,j,k) + T(i,j-1,k)}{(\Delta y)^2} \tag{2.4}$$

$$\frac{\partial^2 T}{\partial z^2} \approx \frac{T(i,j,k+1) - 2T(i,j,k) + T(i,j,k-1)}{(\Delta z)^2} \tag{2.5}$$

Furthermore, it is important to notice that $\Delta x = \Delta y = \Delta z = \delta$, since the mesh is uniformly spaced (as it was shown in Figure 2.1). Using these approximations, the heat equation (2.2) can be approximated and the temperature change can be determined as the following:

$$\begin{aligned}\Delta T \approx \frac{\alpha \Delta t}{\delta^2}(&T(i+1,j,k) - 2T(i,j,k) + T(i-1,j,k) \\ &+ T(i,j+1,k) - 2T(i,j,k) + T(i,j-1,k) \\ &+ T(i,j,k+1) - 2T(i,j,k) + T(i,j,k-1))\end{aligned} \tag{2.6}$$

The forward-time central difference method is relatively simple to implement and is computationally efficient. However, it has stability limitations in terms of the time step size Δt. The time step must be chosen carefully to ensure stability, and an excessively large time step can lead to numerical instabilities. In order to avoid instability issues, the time step size was chosen to satisfy the Courant–Friedrichs–Lewy (CFL) condition, which is a stability criterion used in numerical simulations, including heat conduction simulations.

In the context of heat conduction simulations, the CFL condition takes into account the thermal diffusivity of the material and the grid spacing in each dimension (Equation 2.7). By considering the ratio of the time step to the square of the grid spacing, the condition quantifies the accuracy and stability of the numerical solution.

$$\alpha \Delta t \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2} \right) < C \tag{2.7}$$

The constant C is a threshold limiting the value of time step size. Using a discrete Fourier analysis, one can derive that this threshold is $C = 0.5$, see [17]. Recalling again that $\Delta x = \Delta y = \Delta z = \delta$, the inequality reduces to the form shown in Equation (2.8).

$$\frac{3\alpha \Delta t}{\delta^2} < C \tag{2.8}$$

2.3 Generated datasets

In this section, we present the generated datasets that were created using the previously introduced simulation framework in order to achieve Objective 1.

For the sake of focusing on the main goal which is solving the inverse heat conduction problem with neural network approach, we performed the simulations with only one air bubble in the cube. The target variable is a four dimensional vector containing the parameters of this bubble, that is $y^T = \begin{pmatrix} x_b & y_b & z_b & d_b \end{pmatrix}$. All of them are relative values between 0 and 1, so there are no scaling issues while changing parameter n. x_b, y_b, z_b are the coordinates of the center of the bubble in the coordi-

nate system shown in Figure 2.1, while d_b is the diameter of the bubble.
The input of the network is a tensor with a shape of $(2, n, n)$ containing the temperature measurements recorded in two points of time on the upper plane of the cube, hence the two channels. The intuition behind using multiple channels is to try to capture the rate of temperature change in the upper plane. The two snapshots were taken 5 and 10 seconds after the start of the simulation, since based on experience, in this case, there is no time for the temperature distribution to equalize on the x axis.

The temperature of the heat source was chosen to $T_1 = 400\,^\circ\mathrm{C}$, while $T_2 = 20\,^\circ\mathrm{C}$ was used as room temperature. Note that using values in degrees Celsius is not an error, as temperatures only appear as differences in the equations used. The edge length of the cubes was chosen to $L = 0.1$ m, and we assumed it was cast from copper. This is simply because the thermal diffusivity[1] of copper ($1.11 \times 10^{-4} \frac{\mathrm{m}^2}{\mathrm{s}}$) is an order of magnitude greater than that of air ($1.9 \times 10^{-5} \frac{\mathrm{m}^2}{\mathrm{s}}$), thereby increasing the effect of inclusions on heat conduction.

After performing simulations on a scale of different grids, we decided to use hyperparameter $n_1 = 32$ for Objective 2, that is to develop a neural network architecture that can accurately predict the location and size of the inclusions in the cube. The reason of this parameter setting is that in this case, the computational costs of simulating the heat conduction are relatively low: The number of pixels in the discretized space is 32768 and the simulation takes about 0.42 seconds. For Objective 3, we used $n_2 = 64$ in order to simulate a finer grid for testing purposes. By halving the grid space, the pixel number increased to $n_1^3 \cdot \left(\frac{n_2}{n_1}\right)^3 = 262144$, while the running time of the simulation raised to 38.9 seconds. One can notice that the increase in runtime is not cubic. The main reason of this is that we had to reduce also the time step size Δt from 0.01 to 0.0025 in the second setup to satisfy the stability condition in (2.8).

The samples were generated with a fixed number shown in Table 2.1. The bubble sizes column contains the possible edge length of the bubble in pixels that was drawn for a uniform distribution for each sample. The running times in minutes used for the complete simulations are also shown in the table.

[1] Thermal diffusivity values were taken from the following table: `https://www.engineersedge.com/heat_transfer/thermal_diffusivity_table_13953.htm`

n	# samples	bubble sizes (pixel)	simulation time (min)
32	2048	1,2,3,4,5	14.3
64	64	2,3,4,5,6,7,8,9,10	41.5

Table 2.1: Simulation configurations

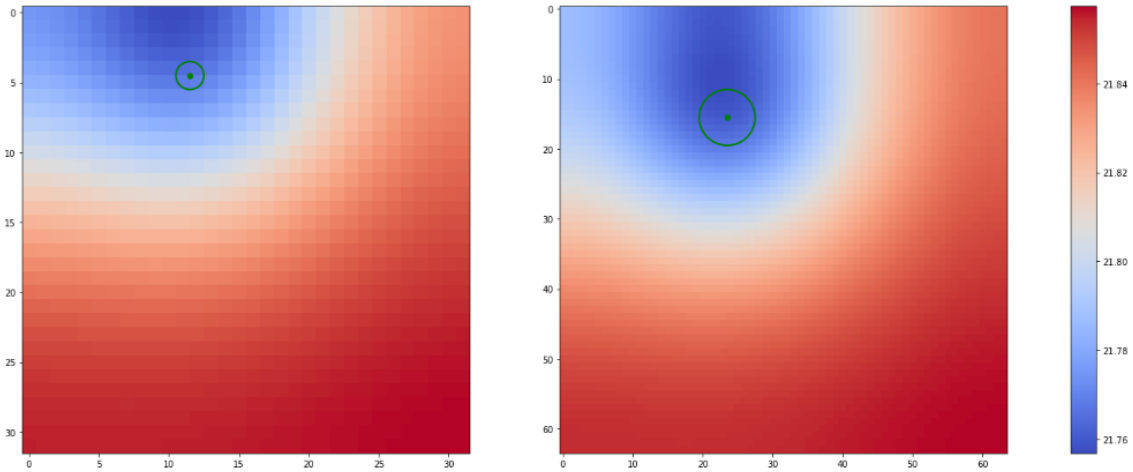

Figure 2.2: Examples of simulated measurements. **Left:** $n = 32$. **Right:** $n = 64$. The green dots indicate the location of the air bubble, while the circles indicate its size.

Chapter 3

Neural network model

In this chapter, we present a number of experiments in order to fulfill Objective 2, i.e. design a neural network architecture which is able the detect the inclusion in the cube. First, the general decisions are introduced regarding the data preprocessing, loss function and learning technique. After that we turn to the architecture design starting with the Multilayer Perceptron model as an initial approach which is followed by the significantly more suitable convolutional approach. The models were trained on the generated data, where $n = 32$, while at the end of the chapter the best models were evaluated on the finer data set ($n = 64$), where further refinement is proposed.

3.1 Data normalization

Based on the results of initial experiments, we found that normalizing the data in each channel separately helps the model to learn and generalize. Therefore, the temperature history X was normalized in each channel as follows.

$$\hat{X}_c = \frac{X_c - \mathrm{E}[X_c]}{\sqrt{\mathrm{Var}[X_c]}} = \frac{X_c - \bar{X}_c}{\sigma_{X_c}} \qquad c \in (1, C), \tag{3.1}$$

where X_c denotes the temperature measurements of the c-th snapshot (c-th channel of X) and C is the channels size of X ($C = 2$ in our case). The expected value $\bar{X}_c$ and standard deviation σ_{X_c} of X_c were calculated over the entire dataset. In this way, the expected value and standard deviation will be 0 and 1 separately for each channel, so the network can only focus on the distribution and the variation of the distribution between channels.

3.2 Loss function

Our network is a regression model that outputs a four dimensional vector $\hat{y}$ containing the position and size of the air bubble in the cube, that is $\hat{y}^T = \begin{pmatrix} \hat{x_b} & \hat{y_b} & \hat{z_b} & \hat{d_b} \end{pmatrix}$. Accordingly, a suitable loss function must be defined to calculate the deviation of the regression. Mean Squared Error (MSE) is a widely used loss function for this type of problems, which actually calculates the squared L_2 norm of the distance vector between the predicted and actual values.

$$L(y, \hat{y}) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} (y_{ij} - \hat{y}_{ij})^2 \tag{3.2}$$

In the equation (3.2), N represents the total number of samples (or batch size in case of mini batches), M equals to the number of dimensions of the target vector y ($M = 4$ in our case), y_{ij} is the actual value for the i-th sample at the j-th position of y, and $\hat{y}_{ij}$ is the associated predicted value.

3.3 Learning algorithm

Backpropagation was chosen as our primary algorithm for computing gradients which is a fundamental algorithm used for training neural networks. It enables the network to learn from input data by iteratively adjusting the weights and biases of each neuron to minimize the overall error. The algorithm calculates the gradients of the network's parameters with respect to the loss function (3.2), facilitating efficient learning. In order to optimize the network's parameters effectively, Stochastic Gradient Descent (SGD) was employed. SGD is an iterative optimization algorithm that updates the weights and biases using a random subset of the training data at each iteration, known as a minibatch. This approach introduces randomness, allowing the network to avoid getting stuck in local minima and converge towards a better overall solution. The choice of minibatch size is a critical design decision. We have selected a minibatch size of 128, which strikes a balance between computational efficiency and convergence speed. A larger minibatch size may consume excessive memory resources and slow down the training process, while a smaller size could lead to noisy gradient estimates and slower convergence.

The parameterization follows the practice in [18]. The learning rate was fixed at 0.01 without any special scheduling technique as our focus is on choosing the most suitable architecture. A momentum of 0.9 was used for better convergence and a weight decay of 0.001 for regularization purposes but Dropout [19] was not used. The training processes were stopped when the validation error plateaued, but the maximum number of epochs was fixed at 1500.

3.4 Network architecture

3.4.1 Activation functions

A crucial aspect of Backpropagation is the choice of activation functions. We have incorporated appropriate activation functions in each layer of the used neural networks. The selection of activation functions can greatly influence the learning process by introducing non-linearities into the model, allowing it to learn complex patterns and relationships within the data. The models were trained using the following activation functions:

$$\mathrm{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.3}$$

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.5}$$

$$\mathrm{LReLU}(x) = \max(0, x) + s \times \min(0, x). \tag{3.6}$$

As revealed by [14], hyperbolic tangent (3.3) can be an effective choice as an activation function to solve the inverse heat conduction problem. Although using Sigmoid activation functions (3.4) may lead to the vanishing gradient problem firstly presented in [15], it has also been tested. In addition, we attempted applying the Sigmoid function to the output layer, since the target vector has a value between 0 and 1, which is the same range as the sigmoid mapping range. Using Rectified Linear Unit (3.5) activation functions avoids the vanishing gradient problem. Furthermore, it has several other advantages such as simplicity, efficiency and improved training

convergence. Also, it can result in sparse activation, meaning that only a subset of neurons in a layer becomes active while the rest remain inactive. This sparsity can help to reduce the computational load and memory requirements, making the network more efficient during both training and inference. However, ReLU can lead to information loss in our case, as the data has been normalized around 0, so half of the data points have a negative value, which may make the learning harder using ReLU. To alleviate this difficulty while retaining the listed benefits, Leaky Rectified Linear Unit activation function (3.6) has also been investigated, which transmits the negative values as well weakened by a constant s.

3.4.2 Multilayer Perceptron approach

The Multilayer Perceptron (MLP) is a fundamental type of feed-forward neural networks, consisting of multiple layers of interconnected artificial neurons, known as perceptrons. It is a versatile and widely used architecture for solving various machine learning tasks, including the inverse heat conduction problem as described in Section 1.2. From our point of view, this architecture was meant to be an initial approach to investigate the basic capabilities of the selected hyperparameters and the neural networks in general to solve the IHCP.

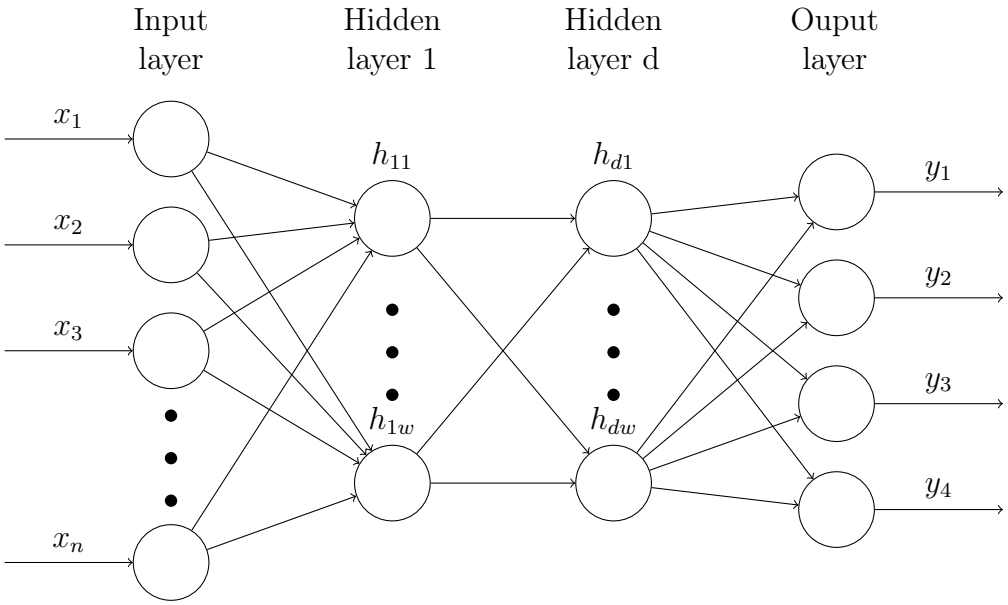

Figure 3.1: Multilayer Perceptron architecture

The MLP architecture is demonstrated in Figure 3.1. In order to feed the input data to the network, it has to be flattened out first. Therefore, the size of the

input layer must be $C \times n^2 = 2 \times 32^2 = 2048$ in our case. The size of the output layer must match with the dimensionality of the target variable y, which is 4. The depth and width of the network, i.e. the number of hidden layers and their size can be chosen arbitrarily, these were also hyperparameters. The activation functions described in Section 3.4.1 were applied on the layers in different configurations. We have also tested to insert Batch Normalization layer after the linear transformation and before the non-linear activation function in each layer of a neural network. Batch Normalization is a technique commonly used in neural networks to improve training stability and accelerate convergence. It addresses the internal covariate shift problem by normalizing the intermediate activations within each mini-batch during training (see details in [20]). It acts as a form of regularization by adding noise to the network through the mini-batch statistics, which can reduce overfitting. Furthermore, Batch Normalization can accelerate the convergence of neural networks by reducing the dependence of the gradients on the scale of the parameters and it helps stabilize the network during training by reducing the effects of internal covariate shift, making the network less sensitive to the initialization and learning rate choices.

network depth	network width	activation function	# params	# epochs	training time (sec/epoch)	MSE
3	128	ReLU	295812	679	0.080	0.0060
2	256	ReLU	591364	533	0.116	0.0061
2	128	ReLU	279300	481	0.076	0.0064
2	256	LReLU	591364	546	0.126	0.0072
3	128	LReLU	295812	682	0.086	0.0075
2	128	LReLU	279300	538	0.076	0.0078
2	128	Tanh	279300	732	0.071	0.0101
3	128	Tanh	295812	720	0.076	0.0101
2	256	Tanh	591364	644	0.111	0.0104
2	128	Sigmoid	279300	1056	0.077	0.0179
2	256	Sigmoid	591364	662	0.120	0.0199
3	128	Sigmoid	295812	355	0.083	0.0297

Table 3.1: Results of Multilayer Perceptron experiments

The dataset was split into training and test datasets using a test set ratio of 0.5. Consequently, the models were trained on a dataset of 1024 samples and evaluated on 1024 samples as well. Table 3.1 shows the results of the performed experiments with different hyperparameters, where the individual configurations records are listed in ascending order of loss (MSE). These results are from training without Batch Normalization and (Sigmoid) output transformation since their application produced

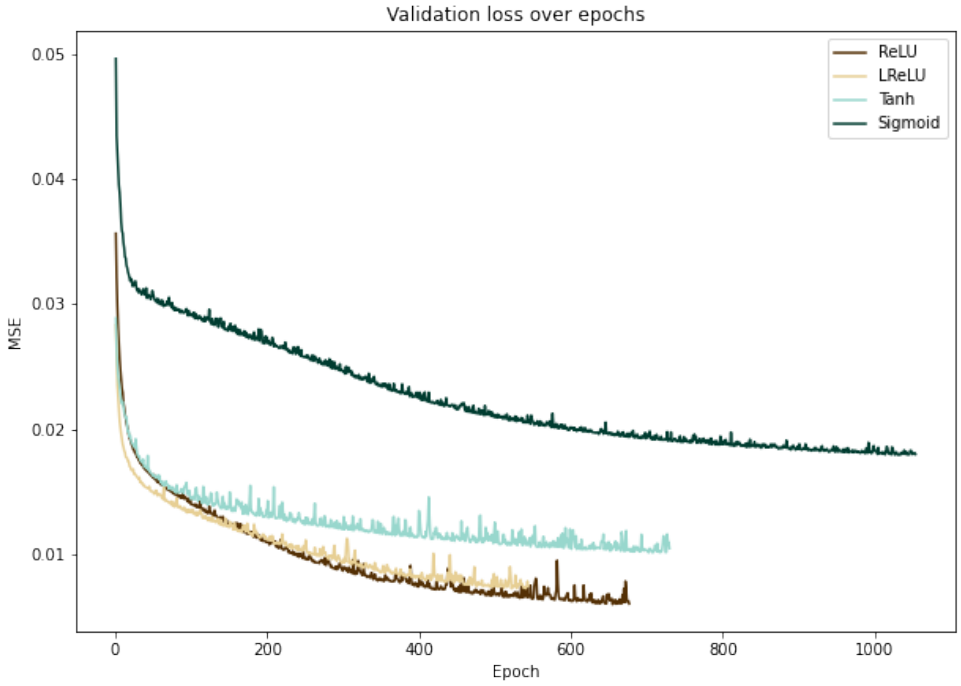

Figure 3.2: Validation losses during the training process of MLP models.

significantly worse results. The table indicates that the Rectified Linear Unit and the Leaky Rectified Linear Unit activation functions are the most suitable choices for this task. Figure 3.2 illustrates the learning curves in terms of validation losses of the best models from each activation function category. The best model (3×128 ReLU) achieved an MSE score of 0.006, this value will serve as a reference during further work.

3.4.3 Convolutional approach

Convolutional Neural Networks (CNNs) are a specialized type of artificial neural networks designed to automatically learn hierarchical representations of visual data. They are specifically designed for processing and analyzing spatial data, such as images or in the present case, the temperature distribution. The convolutional and pooling operations in CNNs allow them to effectively extract and model spatial features, capturing important information about the distribution of temperature values. MLPs, on the other hand, treat the input as a flat vector and may struggle to exploit the spatial structure present in the data. CNNs are capable of automatically learning hierarchical representations of features. In the given task, different levels

of abstraction can be inferred from the temperature history. Lower-level layers of a CNN can capture simple spatial patterns, such as temperature gradients, while higher-level layers can learn more complex features related to the position and size of air bubbles. This hierarchical feature learning enables CNNs to capture both local and global contextual information, improving their ability to predict inclusion parameters accurately. However, convolutional networks are naturally suited to tasks where exact alignment or precise localization is not necessary (mainly in image recognition [21, 22, 23]). In the case of predicting air bubble locations, the positions of the bubbles vary from one observation to another. CNNs can handle these spatial variations by leveraging the pooling layers and shared weights. Furthermore, the translation invariant property of the convolutional networks can be beneficial in predicting the depth and size of the bubble. Nevertheless, Convolutional Neural Networks, combined with techniques like region proposal networks and anchor-based methods, have significantly advanced the state-of-the-art in object detection (see, for instance in [24, 25, 26, 27]).

A typical Convolutional Neural Network architecture consists of the following elements:

1. *Convolutional Layers.* The core building block of CNNs is the convolutional layer. It applies a set of learnable filters (also known as kernels or feature detectors) to the input data, which captures local spatial patterns. Each filter performs a convolution operation by sliding over the input image, computing dot products at each location, and producing feature maps. Multiple filters are typically used to extract different types of features at multiple scales.

2. *Pooling Layers.* Pooling layers are often used in conjunction with convolutional layers. They reduce the spatial dimensions of the feature maps, effectively downsampling the learned representations. The most common pooling operation is max pooling, which selects the maximum value within each pooling window. This process helps to make the representations more invariant to small translations and reduces the computational burden.

3. *Fully Connected Layers.* Following one or more convolutional and pooling layers, CNNs usually incorporate fully connected layers. These layers capture high-level abstractions by combining the features learned from previous lay-

ers. The final fully connected layer(s) typically map the learned representations to the desired output, such as class probabilities in classification tasks.

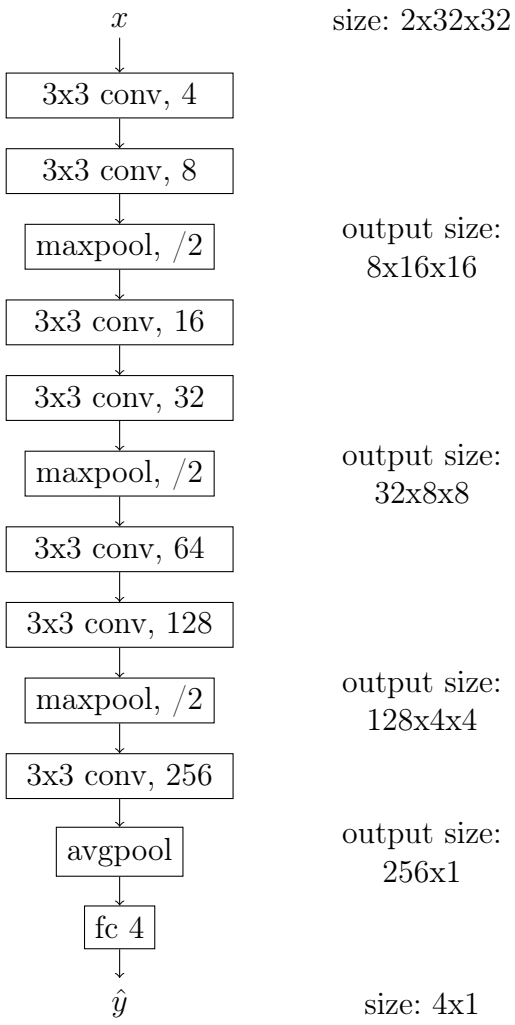

Figure 3.3: Convolutional network architecture

Figure 3.3 demonstrates the architecture of the CNN was used in our work. Every convolutional layer is followed by a Batch Normalization layer and an activation function. In parallel to reduction of spatial dimensions, the number of channels was increased exponentially. After convolutional feature extraction, a global average pooling was applied before the final fully connected regressor. The models were trained using different activation functions listed in Section 3.4.2, except the Sigmoid function since its performance was very poor in the MLP case. All configurations were explored with and without Sigmoid transformation applied on the output, but Batch Normalization was used in each. Table 3.2 contains the results just like in the MLP case. It is important to note that, on average, an order of magnitude smaller

activation function	Sigmoid transform	# params	# epochs	training time (sec/epoch)	MSE
ReLU	Yes	395744	279	3.012	0.00071
LReLU	Yes	395744	238	3.074	0.00076
ReLU	No	395744	167	3.048	0.00085
LReLU	No	395744	193	3.127	0.00088
Tanh	No	395744	97	3.018	0.00108
Tanh	Yes	395744	157	3.063	0.00158

Table 3.2: Results of experiments using convolutional networks

loss was achieved than in the previous case. Although the time required to train the models increased greatly, the number of necessary epoch steps until convergence could be pushed down. In contrast to MLPs, in case of the convolutional architecture, it can be seen that the application of the Sigmoid function on the output layer improved the performance of the models. The architecture using ReLU as activation function featured prominently again achieving the lowest MSE score of 0.00071. As the Figure 3.4 shows, the Tanh model was in the first place during the early steps, but it plateaued quickly. In contrast, the ReLU model was able to learn over 279 epochs, gradually improving.

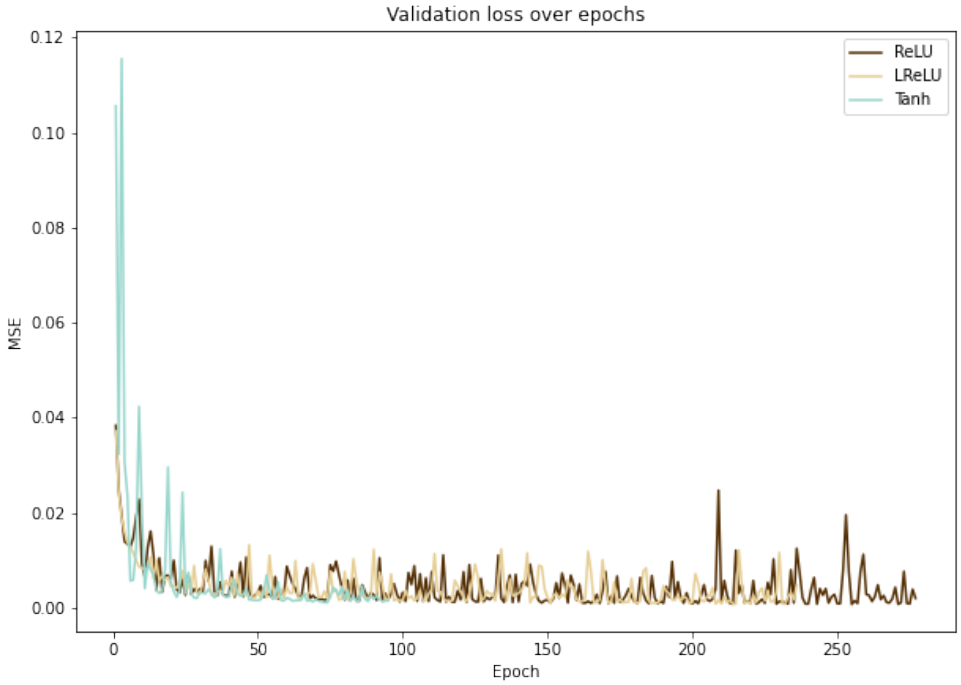

Figure 3.4: Validation losses during the training process of CNN models.

3.5 Performance of the network

We obtained that the best convolution model achieved an MSE of 0.00071. Although we can notice that this value is significantly lower than the average loss of the MLP models, it is uncertain whether this MSE value is satisfactory for our specific case. In order to understand the meaning of this score in terms of inclusion detection, we visualized two predictions having MSE score near this average loss value in Figure 3.5. In this way, it can be seen that this MSE score belongs to predictions which are really close to actual targets. Thus we can mark Object 2 as completed.

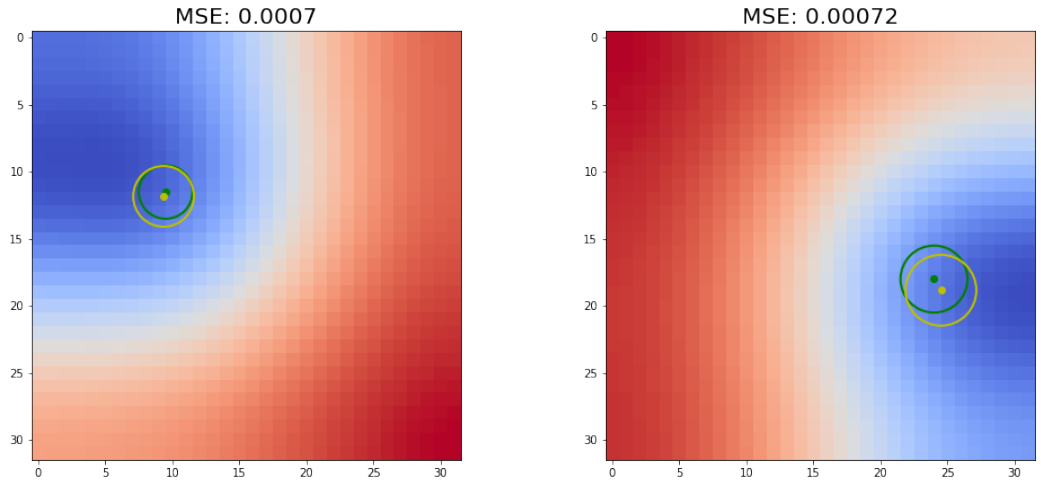

Figure 3.5: Average loss predictions ($n = 32$) **Green:** Ground truth. **Yellow:** Predicted location and size.

Now let us move on the Objective 3 and evaluate our model on the finer dataset, where the temperature history was simulated on a grid with 64 points in each direction. We anticipated favorable outcomes due to two main reasons. On the one hand, Convolutional Neural Networks can handle inputs with objects or patterns at different scales. Whether it is detecting small objects or capturing large-scale structures, CNNs can effectively learn and recognize features across various scales without explicitly requiring prior knowledge or manual scaling of the input data. On the other hand, the target variable has been normalized with the intention of making the prediction task scale invariant. However, despite these efforts, the model failed to achieve satisfactory results. The average MSE obtained is 0.093, which is two orders of magnitude higher than results obtained on the coarser grid. Two examples

of these predictions are visualized in Figure 3.6, in order to illustrate that this loss value corresponds to an unreliable prediction. In case of the coarser dataset, the model was able to accurately predict both the size and position of the air bubble, whereas, this is still no longer valid. These findings highlight the importance of some further refinement procedure.

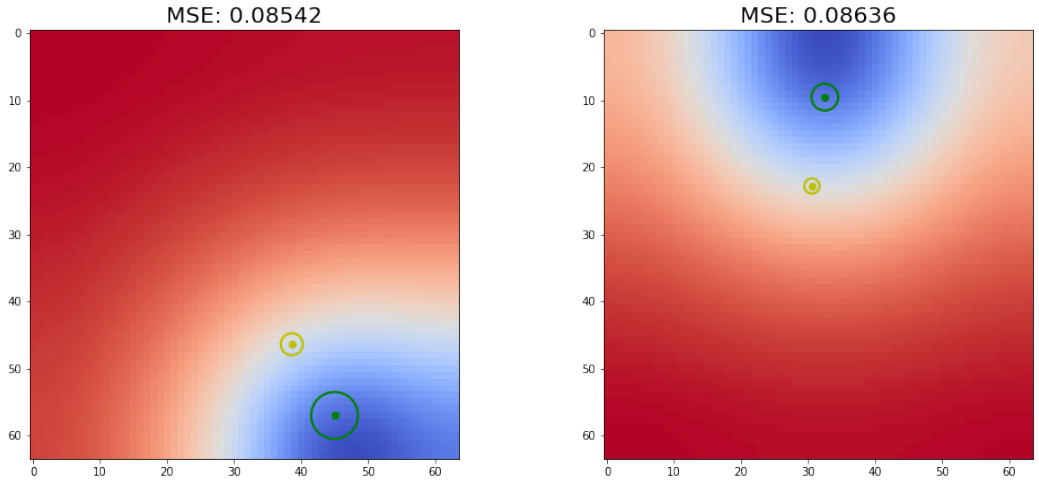

Figure 3.6: Average loss predictions on finer grid ($n = 64$) **Green:** Ground truth. **Yellow:** Predicted location and size.

Chapter 4

Multigrid motivated refinement

Based on an earlier assumption, if we train a Convolutional Neural Network on a coarser scale, it can transfer its knowledge to a finer scale. However, as it was described in 3.5, the convolutional model did not produce the test results that met these expectations. A possible reason for this is that since the network was trained on a limited amount of coarse data and then tested on finer data, there is a chance of overfitting. The network might struggle to generalize well to the higher-resolution data due to the limited variability in the training set. In such cases, additional training data with fine-grained measurements or regularization techniques can help to mitigate overfitting and improve performance. Although this is not a viable option as the finer data can only be used for testing purposes based on our definition, a reasonable reduction method can be applied to finer data samples before they enter the network compressing the input to the size used by the model. In this way, even reducing the size of the input with average pooling improves the performance of the model resulting an MSE score of 0.059. At this point, the Multigrid method [28] becomes applicable in order to further improve the performance of the model on finer data. It aims to solve Partial Differential Equations reformulating the system as subproblems at multiple scales, where each subproblem is responsible for the residual solution between a coarser and a finer scale. In this chapter, we seek for a new multigrid motivated neural network architecture that extends the previously designed convolutional model outperforming the simple average pooling restriction operator.

4.1 Encoder-based restriction operator

Autoencoders [16] are a class of neural networks commonly used in unsupervised learning and dimensionality reduction tasks. They consist of an encoder and a decoder, which work together to learn a compressed representation of the input data. The primary goal of an autoencoder is to reconstruct the input data as accurately as possible.

The encoder component of an autoencoder takes the input data and maps it to a lower-dimensional latent space representation. This mapping process involves a series of hidden layers that progressively reduce the dimensions of the input. The encoder tries to capture the most important features and patterns of the data, compressing it into a condensed representation.

The decoder component then takes this compressed representation from the encoder and attempts to reconstruct the original input data. Similar to the encoder, the decoder consists of a series of hidden layers that gradually increase the dimensionality until the output matches the input dimensions. The reconstructed output is compared to the original input, and the network is trained to minimize the reconstruction error, typically using a loss function such as mean squared error we used earlier.

The restriction operator is a key component in multigrid methods, used to coarsen the grid and transfer information from a fine grid level to a coarser grid level. Autoencoders, with their ability to learn compressed representations and extract important features, can potentially be leveraged to learn an approximation of the restriction operator. Figure 4.1 demonstrates the desired architecture corresponding to the new approach. The finer data x with a shape of $2 \times 64 \times 64$ is first mapped into the latent space representation z which has the same shape $2 \times 32 \times 32$ as the coarse training data had. Therefore, passing this tensor z to the pretrained convolutional network we can expect better performance.

4.1.1 Training process

In order to train our encoder (Fig. 4.1) to approximate the restriction operator using only the coarser data, there are some additional issues to consider. Since the samples from the training dataset have the shape $2 \times 32 \times 32$ and the encoder halves the spatial sizes, the input of the convolutional network has a shape of $2 \times 16 \times 16$ during training. This suggests that the encoder and the convolutional regressor should

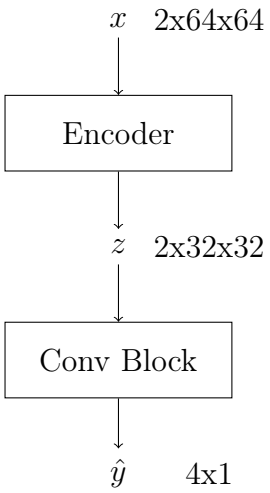

Figure 4.1: High-level network architecture

be trained separately, seeing that the performance of the convolutional model decreases when inferring samples of different size than the ones the model was trained on. For this reason, the previously trained convolutional network was used in this architecture. As Figure 4.2 illustrates, a decoder component was added to the training architecture with the objective of reconstructing the input data x from the latent representation z, thereby resulting a reconstruction tensor $\hat{x}$. This tensor is used to penalize the model if it produces inappropriate latent representations (which do not store sufficient information of the input for reconstruction) using the reconstruction loss $R(x, \hat{x})$ defined in (4.1), where N is the batch size and M is the size of the input tensor x. In this way, our final loss function $\hat{L}$ takes the form shown in (4.2), where α was introduced as a hyperparameter of the training. This is a crucial point in our separate training to avoid overfitting since without this penalty, the encoder could learn substantially different representations from the one expected by the convolutional network.

$$R(x, \hat{x}) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} (x_{ij} - \hat{x}_{ij})^2 \tag{4.1}$$

$$\hat{L}(x, y, \hat{x}, \hat{y}) = L(y, \hat{y}) + \alpha R(x, \hat{x}) \tag{4.2}$$

The training process of the final neural network can be divided into the following steps:

1. *Training the convolutional block.* This step applies to the convolutional network with an input shape of $2 \times 32 \times 32$ which was described in Chapter 3.

2. *Training the autoencoder.* For this step, the architecture in Figure 4.2 was used with randomly initalized weights in the entire network. Due to the use of the loss function $\hat{L}$ in (4.2), the encoder was trained to compress x into a latent space, from where it can be reconstructed by the decoder and the convolutional network can produce accurate predictions.

After these training steps, we took the encoder out of the training architecture and placed it directly before the pretrained convolutional block. That is the decoder component and the newly trained convolutional block have been discarded. In this way, the resulting model became the final inference-time model (see in Figure 4.1).

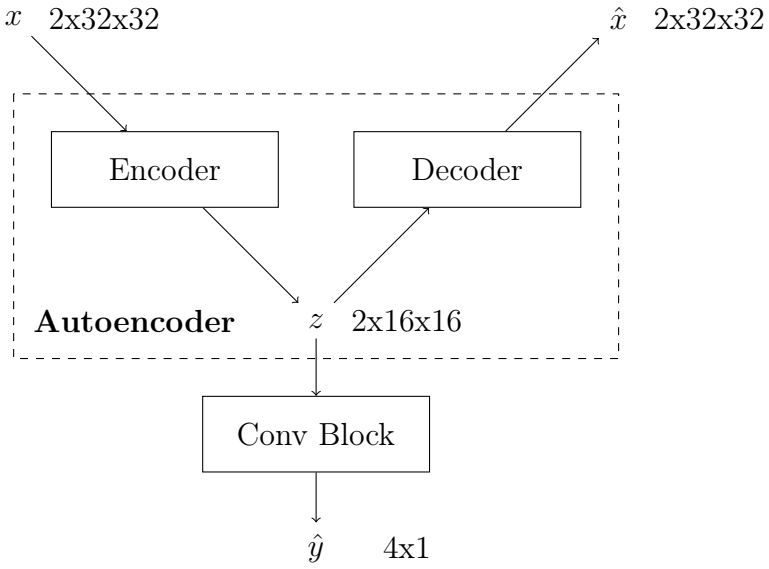

Figure 4.2: Autoencoder-based network architecture used for training

4.1.2 Residual autoencoder architecture

The high-level architecture of the autoencoder is shown in Figure 4.2; now we present the detailed architecture of its components as well. As it was mentioned earlier, we were inspired by the Multigrid method when designing the network. Accordingly, the encoder must perform a restriction on the input to a coarser scale at which the convolution block can detect inclusions. In general, the restriction operation aims to retain the most important information from the fine grid while effectively reducing the problem size on the coarser grid. A common approach to this is that the average of neighboring grid points on the fine grid gives the corresponding point of the coarse grid. After the restriction operation, the Multigrid method can

proceed to solve the system of equations on the coarser grid, typically using a few iterations of a relaxation method. The coarser grid solution represents an approximation to the error at the corresponding scale, and it can then be transferred back to the finer grid through a process called interpolation or prolongation.

In our autoencoder-based approach, in addition to serving as a restriction operator with averaging, the encoder is also responsible for correcting the resulting error. As shown in Figure 4.3, the encoder has two branches. The right-hand branch executes average pooling, halving the spatial sizes uniformly, while the other branch represents the approximation error using three convolutional layers. Rectified Linear Unit activation function was applied after each convolutional layer. The coarser representation z is calculated as the sum of these two results. Using this representation as input, our convolutional network generates the coarser grid solution, resulting in the final output $\hat{y}$ (Fig. 4.2).

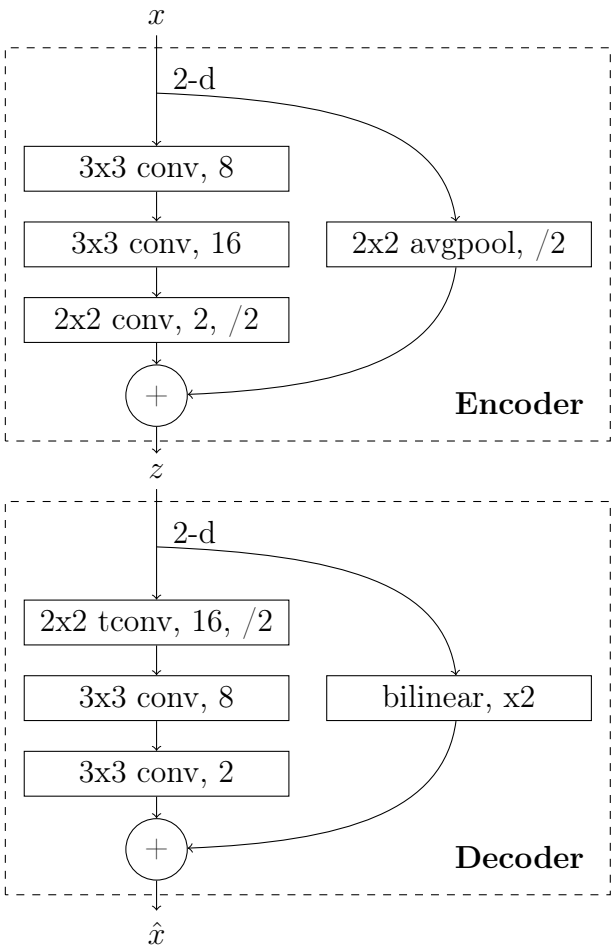

Figure 4.3: Residual autoencoder architecture

The decoder component is responsible for reconstructing the input for training

purposes. This reconstruction is also achieved using a residual connection, where the result $\hat{x}$ is obtained by adding a bilinear interpolation and an error term. For the error term, we utilize transposed convolution (denoted with tconv) to restore the spatial sizes. We also tried the case where the decoder is simply an interpolation operator, adding extra information to the system. This reduced the number of parameters, thus the encoder can be trained faster, and overfitting can be avoided. This type of decoder can be beneficial especially in the case of small α, because we are not interested in the reconstruction being accurate.

A denoising autoencoder was used in [13] in order to encode the temperature measurements into a latent space reducing the dimension and noise on the experimental data. These reduced data formed the input of their MLP network. Denoising autoencoders can be viewed either as a regularization option, or as robust autoencoders which can be used for error correction [29]. However, in our case, the size of the latent representation is significantly smaller compared to the input size since our latent space represents the coarser observations. Therefore, adding input noise could prevent the model from learning, hence we ignored this option now. Furthermore, the larger the air bubble and the deeper it is located, the lower is the standard deviation of temperature distribution on the upper plane. So while in one case, the addition of a random Gaussian noise $e \sim N(0, 0.1)$ results in the expected effect (Fig. 4.4 right side), it can also lead to completely noisy distribution in case of samples with low standard deviation (Fig. 4.4 left side).

4.2 Performance of the network

As it was seen earlier, the best convolutional network achieved an MSE of 0.00071 on the coarser data, while it yielded unsatisfactory outcomes on the finer data as the average MSE obtained was 0.093 in that case. We also found that if the spatial sizes are halved using average pooling before seeding the samples into the convolutional network, this loss value can be reduced to 0.059. In this chapter, we introduced a multigrid motivated refinement in order to further reduce this score by placing an additional encoder before the pretrained convolutional block. Tables 4.1 and 4.2 contain the result corresponding to different training approaches. The first one belongs to the autoencoder illustrated in Figure 4.2, where both the encoder and decoder components were based on residual connections. Different rows represent training re-

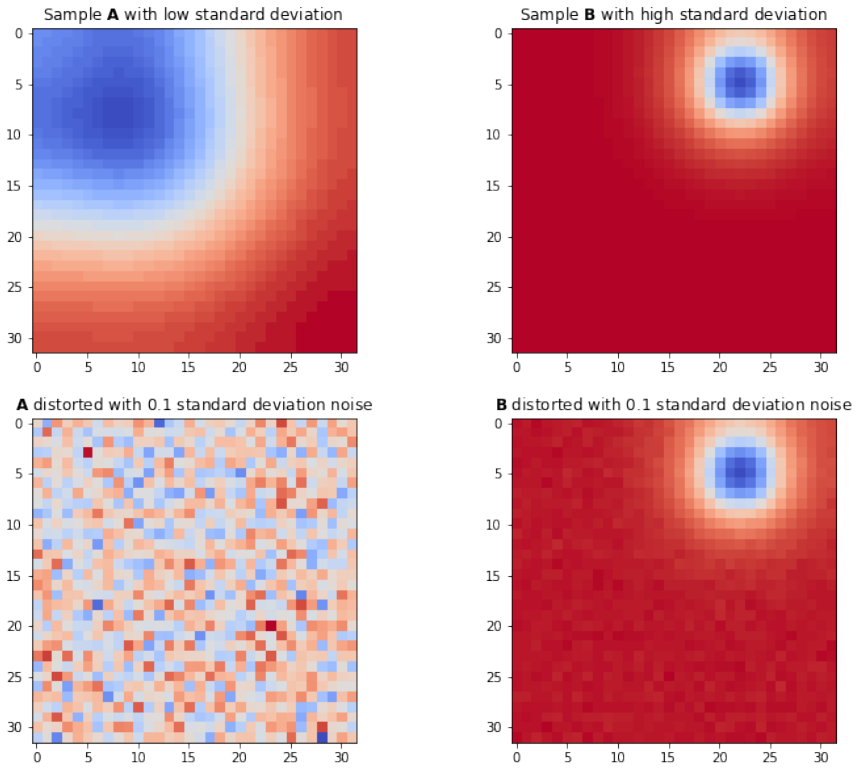

Figure 4.4: Effect of added noise on samples with different standard deviations. **Left:** The sample with low standard deviation becomes completely noisy. **Right:** A sample with high standard deviation is hardly affected by the added noise.

sults using different vales for the hyperparameter α in the loss function (4.2), where the far right column shows the MSE scores obtained on the finer dataset. Based on these results, it can be stated that this autoencoder-based technique successfully improved the performance of the model. However, one can notice that it can be even more accurate using simple bilinear interpolation as decoder during training. This choice also resulted in fewer and faster epochs as we expected.

The weights of the model were optimized subject to the loss function $\hat{L}$ in (4.2). Figure 4.5 shows an example of the evolution of obtained loss values during training. The brown line chart represents the validation losses after each epoch. Also, the test losses were obtained by evaluating the model shown in Figure 4.1 on the finer dataset which is illustrated by the green curve. This model consists of the encoder in training and the pretrained convolutional block. The chart demonstrates that the autoencoder-based model can generalize to finer data.

α	# epochs	training time (sec/epoch)	test loss (L)
0.01	101	4.037	0.0123
1.00	115	4.165	0.0207
0.10	105	4.010	0.0294
0.20	87	4.090	0.0341
0.05	108	3.967	0.0381

Table 4.1: Results of autoencoder-based experiments with residual decoder

α	# epochs	training time (sec/epoch)	test loss (L)
0.10	91	3.230	0.0021
0.05	113	3.085	0.0048
0.01	45	3.397	0.0119
0.20	42	3.177	0.0128
1.00	47	3.266	0.0241

Table 4.2: Results of autoencoder-based experiments with simple decoder

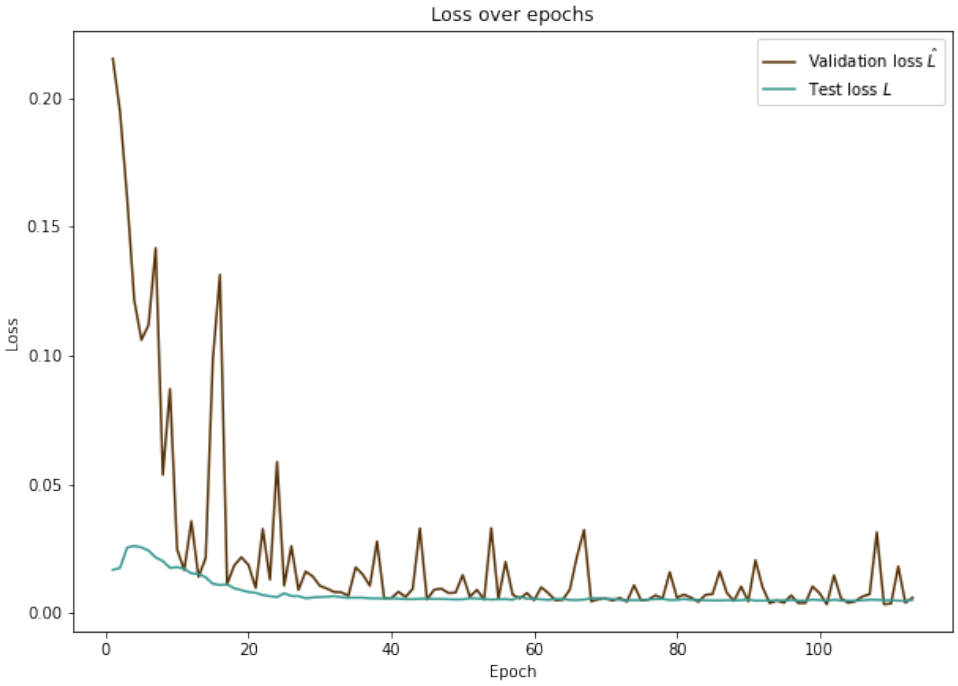

Figure 4.5: Obtained loss values during training in case of simple decoder and an α value of 0.05. **Brown:** The loss $\hat{L}$ of the training network (Fig. 4.2) on the validation data. **Green:** The loss L of the final network (Fig. 4.1) obtained on the finer test data.

Chapter 5

Discussion

In this study, we sought to design a suitable Neural Network architecture that can solve the Inverse Heat Conduction Problem on different scales. Through the analogy of finding inclusions in cast metal based on the measured temperature history, different network architectures were investigated. This chapter is dedicated to discuss the interpretation and implications of the obtained results, the strengths and limitations of the study and our suggestions for further research.

5.1 Interpretation and implications of the results

It was presented that a certain prediction accuracy is achievable even with simple Multilayer Perceptron architectures in solving the IHCP. Although these results were not satisfactory, the experiment was perfect to obtain a hands-on idea of what the neural network approach is capable of, as well as an initial intuition for the impact of different activation functions on the optimization process. The results in Table 3.1 show that the Rectified Linear Unit and the Leaky Rectified Linear Unit are the most suitable activation functions for our task. By using them, the model was able to gradually learn to achieve the best performance. In contrast, the Sigmoid activation function led to slow learning and poor performance, possibly due to the vanishing gradient problem.

The convolutional architecture illustrated in Figure 3.3 managed to produce an order of magnitude smaller loss values than the Multilayer Perceptrons did on average. The best model achieved an MSE score of 0.00071 on the coarser validation dataset, which in our case means quite accurate predictions as demonstrated in

Figure 3.5. These results suggest that the IHCP can be adequately approximated by convolutional networks. However, the network trained on the coarser dataset was unable to transfer its knowledge at a finer scale, as it achieved a significantly higher loss on the finer dataset. This also led to completely unreliable prediction at the finer scale as shown in Figure 3.6. Although we have carefully normalized both the input and output of the network in order to support prediction at different scales, and CNNs are naturally able to handle inputs with different spatial dimensions properly, the results do not meet our expectations. This result probably indicates overfitting for the limited amount of coarser data.

Finally, we designed and trained an encoder component which reduces the spatial sizes of the finer data so that its output has the same size as the coarser samples on which the convolutional network was trained (Fig. 4.1). The architecture is inspired by the Multigrid method, so the encoder was designed to accomplish a residual connection that sums the downsized input and the predicted error term (4.3). Two autoencoder architectures were investigated, both of which were able to achieve a lower loss value than the plain convolutional network did, demonstrating the capability of the introduced approach in finer-scale generalization. Based on the presented results, the goals stated at the beginning of the thesis can be considered fulfilled.

5.2 Discussion of the strengths and limitations of the study

This study has two major strengths, starting with the introduced convolutional framework which aims to solve the Inverse Heat Conduction Problem. The data simulation process was designed to generate 3-dimensional data containing planar temperature measurements on each channel at different times. This data structure is suitable for processing by Convolutional Neural Networks as opposed to the approaches in other studies were mentioned in Chapter 1.2. Additionally, a specific convolutional architecture was presented in the study that provides fairly accurate predictions of the location and size of the inclusions. The other main strength of the study is the autoencoder-based network architecture and its training approach. We experienced that our multigrid-motivated network could generalize not only to the validation data of the same scale the model trained on, but also to a finer scale as

the loss values demonstrated in Figure 4.5.

However, our study has some limitations as well. Regarding to the training data and its simulation process, the models were trained on synthetic data without artificially added noise, and some constraints were applied (such as there is exactly one inclusion in each metal cube). Furthermore, a significant amount of discretization was used in order to reduce the computational cost of the simulation. Therefore, the trained models cannot be directly applied on real experimental temperature histories. Rather, they show promise for the solution of inverse problems.

5.3 Suggestions for future research

Although promising results were presented in this study, there is always room for further research. Since the introduced neural network is based on convolutional layers which are naturally suited to tasks where exact localization is not necessary, replacing them with Locally Connected layers could be a reasonable idea. It would lead to increased number of parameters, hence slower training and inference. However, one can notice that the main convolutional network used in this study has only 395744 parameters, which is only a fraction of the parameters of the state-of-the-art convolutional networks used in many areas of industry. It would also be interesting to investigate what kind of result the presented architecture produces in case of arbitrary number of inclusions. This probably requires more training data and more measurements on each sample, i.e. an increased number of channels. We could see that the autoencoder-based neural network achieved good results at the finer scale. However, there is still an order of magnitude difference compared to the results produced by the model on coarser data. This also suggests further research in terms of investigating additional architectures and training methods.

Chapter 6

Summary

This study focused on designing a Neural Network architecture to solve the Inverse Heat Conduction Problem (IHCP) on different scales. We explored various network architectures using the analogy of finding inclusions in cast metal based on temperature measurements. For this, as a first step, we generated data of different fineness for training and testing purposes by simulating heat conduction with predefined boundary conditions and inclusions of different locations and sizes in the metal cube.

The design of the appropriate neural network architecture included the investigation of various activation functions and regularization techniques. The results of initial experiments showed that even simple Multilayer Perceptron architectures can achieve a certain level of prediction accuracy in solving the IHCP. However, convolutional architectures demonstrated significantly lower loss values compared to the Multilayer Perceptrons on average. The convolutional models achieved accurate predictions on the coarser validation dataset, indicating the adequacy of convolutional networks for approximating the IHCP. The initial assumption was that the CNN's knowledge on the coarse scale would transfer to the finer scale, but this was not the case. The limited variability in the training data and the chance of overfitting led to poor generalization to higher-resolution data.

To address this issue, we proposed the use of a multigrid motivated approach. The goal was to develop a new neural network architecture that could outperform the simple average pooling restriction operator. Autoencoders, which are neural networks commonly used for unsupervised learning and dimensionality reduction, were employed to learn an approximation of the restriction operator. The proposed

autoencoder-based network consisted of an encoder, a decoder and a convolutional regressor. The encoder served as the restriction operator compressing the input data into a lower-dimensional latent space representation, symbolizing the coarser scale. While the decoder aimed to reconstruct the original input from the latent representation. This compressed representation was also passed to a pretrained convolutional network for improved performance on finer data. The training process involved separate training of the convolutional block and the autoencoder. A new loss function was introduced which penalized the encoder if it produced latent representations that did not store enough information for accurate reconstruction. After training, the encoder was integrated into the final inference-time model. The resulting architecture improved the performance of the model on finer data by leveraging the multigrid-inspired restriction operator provided by the encoder.

In conclusion, the study introduced a novel approach using autoencoder-based restriction operators to enhance the performance of CNNs on finer-scale data. By leveraging the capabilities of autoencoders to learn compressed representations and extract important features, the proposed architecture improved the model's generalization and achieved better results on higher-resolution data. This research contributes to the field of deep learning and provides insights into addressing the challenges of transferring knowledge across scales in neural networks.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Ferenc Izsák, for his invaluable guidance, support and contributions throughout this research project. His expertise and insights were instrumental in shaping the direction of my work and pushing me to achieve higher standards.

Bibliography

[1] Mohammed Hussein, Daniel Lesnic, and Mansur Ismailov. "An inverse problem of finding the time-dependent diffusion coefficient from an integral condition". In: *Mathematical Methods in the Applied Sciences* 39 (Mar. 2015). DOI: 10.1002/mma.3482.

[2] Liu Yang, Jian-Ning Yu, and Zui-Cha Deng. "An inverse problem of identifying the coefficient of parabolic equation". In: *Applied Mathematical Modelling* 32.10 (2008), pp. 1984–1995. ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2007.06.025. URL: https://www.sciencedirect.com/science/article/pii/S0307904X07001540.

[3] A. Hazanee and D. Lesnic. "Determination of a time-dependent heat source from nonlocal boundary conditions". In: *Engineering Analysis with Boundary Elements* 37 (June 2013), 936–956. DOI: 10.1016/j.enganabound.2013.03.003.

[4] Victor Isakov and Stefan Kindermann. "Identification of the diffusion coefficient in a one-dimensional parabolic equation". In: *Inverse Problems* 16.3 (June 2000), p. 665.

[5] Zui-Cha Deng et al. "Identifying the diffusion coefficient by optimization from the final observation". In: *Applied Mathematics and Computation* 219.9 (2013), pp. 4410–4422. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2012.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0096300312010612.

[6] Ley Chen et al. "Simulation and Experimental Study of Inverse Heat Conduction Problem". In: *Advanced Materials Research* 233-235 (2011), pp. 2820–2823. DOI: 10.4028/www.scientific.net/AMR.233-235.2820.

[7] Krzysztof Grysa. "Inverse Heat Conduction Problems". In: *Heat Conduction - Basic Research*. InTech, 2011. DOI: 10.5772/26575.

[8] Zhaoxing Li and Zhiliang Deng. "A total variation regularization method for an inverse problem of recovering an unknown diffusion coefficient in a parabolic equation". In: *Inverse Problems in Science and Engineering* 28 (Feb. 2020), pp. 1–21. DOI: 10.1080/17415977.2020.1725502.

[9] Jiri Krejsa et al. "Assessment of strategies and potential for neural networks in the inverse heat conduction problem". In: *Inverse Problems in Engineering - INVERSE PROBL ENG* 7 (June 1999), pp. 197–213. DOI: 10.1080/174159799088027694.

[10] Obed Cortés-Aburto et al. "Artificial Neural Networks for Inverse Heat Transfer Problems". In: Oct. 2007, pp. 198 –201. ISBN: 978-0-7695-2974-5. DOI: 10.1109/CERMA.2007.4367685.

[11] S. Deng and Y. Hwang. "Applying neural networks to the solution of forward and inverse heat conduction problems". In: *International Journal of Heat and Mass Transfer* 49.25 (2006), pp. 4732–4750. ISSN: 0017-9310. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2006.06.009. URL: https://www.sciencedirect.com/science/article/pii/S0017931006003711.

[12] Fabio Mikki et al. "A Neural Network Approach In A Backward Heat Conduction Problem". In: Mar. 2016, pp. 019–024. DOI: 10.21528/CBRN1999-008.

[13] Lucas Jardim et al. "Contact Failure Identification in Multilayered Media via Artificial Neural Networks and Autoencoders". In: *Anais da Academia Brasileira de Ciências* 94 (Aug. 2022). DOI: 10.1590/0001-3765202220211577.

[14] Weijia Qian et al. "Physics-informed neural network for inverse heat conduction problem". In: *Heat Transfer Research* (Oct. 2022). DOI: 10.1615/HeatTransRes.2022042173.

[15] Razvan Pascanu, Tomas Mikolov, and Y. Bengio. "On the difficulty of training Recurrent Neural Networks". In: *30th International Conference on Machine Learning, ICML 2013* (Nov. 2012).

[16] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *CoRR* abs/2003.05991 (2020). arXiv: 2003.05991. URL: https://arxiv.org/abs/2003.05991.

[17] J.W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*. Texts in Applied Mathematics. Springer New York, 2013. ISBN: 9781489972781. URL: https://books.google.hu/books?id=83v1BwAAQBAJ.

[18] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.

[19] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[20] Christian Szegedy and Sergey Ioffe. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[21] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[22] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[23] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[24] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. "Deep Neural Networks for Object Detection". In: Jan. 2013, pp. 1–9.

[25] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[26] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[27] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[28] William Briggs, Van Henson, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. Jan. 2000. ISBN: 978-0-89871-462-3.

[29] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: Jan. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.