

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

An Artificial Intelligence framework to surrogate standard FEA in linear elasticity

Guilherme Vítor Gonçalves Ribeiro



Master's in Mechanical Engineering

Supervisor: Professor Doutor Jorge Américo Oliveira Pinto Belinha

Co-Supervisor: Professor Doutor Renato Manuel Natal Jorge

Co-Supervisor: Professora Ana Isabel Lopes Pais

October 1, 2024

An Artificial Intelligence framework to surrogate standard FEA in linear elasticity

Guilherme Vítor Gonçalves Ribeiro

Master's in Mechanical Engineering

October 1, 2024

Abstract

The prediction of mechanical behaviour in Solid Mechanics is crucial for many engineering applications, yet traditional methods such as the Finite Element Method (FEM) are computationally expensive.

This study aims to develop an Artificial Neural Network (ANN) as a surrogate model for FEM to efficiently predict the mechanical behaviour of well known Solid Mechanics benchmarks.

The research involved training ANNs on data generated from FEM simulations for a two dimensional plate with a central hole and a two dimensional plate with four holes, with both cases being subjected to a distributed load. The study explored the impact of outliers on the data, the optimal hyperparameters to enhance the model performance and assessed the accuracy of the model in predicting the desired outputs.

The findings revealed that the neural network can effectively predict the mechanical behaviour of these two cases with very similar results obtained from the FEM solution. However, the accuracy of the predictions is influenced by the presence of outliers, with the ANN performing better when trained on data without outliers. Additionally, the study analysed the influence of hyperparameters on the model's performance and found that a network trained on integration points can accurately predict outcomes at the nodes.

This study demonstrates the potential of using ANNs as efficient and accurate surrogates for FEM in solid mechanics, suggesting that similar models could be applied to a wider range of problems.

Keywords: Artificial Neural Network, Finite Element Method, Solid Mechanics, Hyperparameter optimization, Outliers

Resumo

A previsão do comportamento mecânico em Mecânica dos Sólidos é crucial para muitas aplicações de engenharia, mas os métodos tradicionais, como o Método dos Elementos Finitos (MEF), são computacionalmente dispendiosos.

Este estudo tem como objetivo desenvolver uma Rede Neuronal Artificial (RNA) como um modelo substituto do MEF para prever eficientemente o comportamento mecânico de modelos de Mecânica dos Sólidos bem conhecidos.

A investigação envolveu o treino de RNAs em dados gerados a partir de simulações de MEF para uma placa bidimensional com um furo central e uma placa bidimensional com quatro furos, estando ambos os casos sujeitos a uma carga distribuída. O estudo explorou o impacto dos *outliers* nos dados, os hiperparâmetros ótimos para melhorar o desempenho do modelo e avaliou a precisão do modelo na previsão dos resultados desejados.

Os resultados revelaram que a rede neuronal consegue prever com eficácia o comportamento mecânico destes dois casos com resultados muito semelhantes aos obtidos com a solução do MEF. No entanto, a exatidão das previsões é influenciada pela presença de valores anormais, tendo a RNA um melhor desempenho quando treinada com dados sem *outliers*. Além disso, o estudo analisou a influência dos hiperparâmetros no desempenho do modelo e concluiu que uma rede treinada em pontos de integração prevê com exatidão os resultados nos nós.

Este estudo demonstra o potencial da utilização de RNAs como substitutos eficientes e precisos do MEF em mecânica dos sólidos, sugerindo que modelos semelhantes podem ser aplicados a uma gama mais vasta de problemas.

Palavras chave: Redes Neurais Artificiais, Método dos Elementos Finitos, Mecânica dos Sólidos, Otimização de Hiperparâmetros, *Outliers*

Acknowledgments/Agradecimentos

Primeiramente, gostaria de expressar a minha gratidão ao meu orientador, Professor Doutor Jorge Américo Pinto Belinha, por todo o apoio e orientação prestados durante o semestre. O seu compromisso com o meu desenvolvimento académico foi essencial para o sucesso deste trabalho. A sua sabedoria e paciência deram-me o suporte necessário para alcançar todos os objetivos propostos.

Os meus maiores agradecimentos à Professora Ana Isabel Lopes Pais por me ter acompanhado incansavelmente durante todo o semestre. A sua paciência, preocupação e dedicação foram inestimáveis para o sucesso desta investigação. Agradeço todo o apoio técnico e académico, assim como todas as orientações que contribuíram para a qualidade deste trabalho.

Aos meus amigos por estarem sempre ao meu lado e pela fonte de inspiração que me motivou a querer sempre ser e fazer melhor. A vossa amizade tornou esta jornada mais leve e cheia de momentos que levarei para a vida.

Não posso deixar de agradecer à minha família todo o apoio e carinho que sempre me proporcionaram. Em especial, aos meus irmãos e à minha avó, que mesmo me chateando de vez em quando foram uma presença constante.

À Inês, por estar sempre do meu lado, me apoiar e me aturar nos momentos mais difíceis. Todo o teu encorajamento foi indispensável para que eu pudesse continuar nesta caminhada.

Ao meu pai e à minha mãe agradeço imensamente todas as condições que sempre me proporcionaram, pelo apoio e incentivo ao longo da minha vida académica, e não só. Vocês são o alicerce sobre o qual construí este trabalho e por isso este sucesso também é vosso.

A todos, o meu mais sincero obrigado.

Guilherme Ribeiro

*“Once you know what failure feels like,
determination chases success”*

Kobe Bryant

Contents

1	Introduction	1
1.1	Framework and Motivation	1
1.2	Objectives	2
1.3	Dissertation Structure	3
2	Theoretical Background	4
2.1	Artificial Intelligence	4
2.1.1	Machine Learning	4
2.1.2	Deep Learning	5
2.2	Artificial Neural Networks (ANN)	6
2.2.1	Optimization Algorithms	9
2.2.2	Learning Rate	12
2.3	Solid Mechanics Fundamentals	14
2.3.1	Stress	14
2.3.2	Strain	14
2.3.3	Constitutive Equations	16
2.3.4	Equilibrium Equations	17
2.4	Equations for two-dimensional solids	17
2.4.1	Plane Stress	17
2.4.2	Plane Strain	18
2.5	Galerkin Weak Form	19
2.6	Finite Element Procedure	22
3	State of the Art	27
4	Plate with a central hole	34
4.1	Problem Description	34
4.2	Methodology	36
4.2.1	FEM Analysis	36
4.2.2	ANN Training	37
4.3	Results	41
4.3.1	Node Displacements	41
4.3.2	Stress at Integration Points	52
4.3.3	Strain at Integration Points	64
4.3.4	Stress at Nodes	76
4.3.5	Strain at Nodes	79
4.4	Discussion	82

5	Plate with four identical holes	84
5.1	Problem Description	84
5.2	Methodology	85
5.2.1	FEM Analysis	85
5.2.2	ANN Training	86
5.3	Results	87
5.3.1	Node Displacements	87
5.3.2	Stress at Integration Points	93
5.3.3	Strain at Integration Points	100
5.3.4	Stress at Nodes	107
5.3.5	Strain at Nodes	110
5.4	Discussion	113
6	Conclusions	114
6.1	Study Limitations	115
6.2	Fulfillment of objectives	115
6.3	Future work	116
	References	117
A	Appendix	121
A.1	Annex 1	121
A.2	Annex 2	122
A.3	Annex 3	128
A.4	Annex 4	132
A.5	Annex 5	137
A.6	Annex 6	138
A.7	Annex 7	141
A.8	Annex 8	142
A.9	Annex 9	146

List of Figures

2.1	Subsets of AI	5
2.2	Types of artificial neural networks	6
2.3	Example of an ANN	7
2.4	Activation Functions: Sigmoid, Tanh and ReLU	8
2.5	Small Learning Rate	13
2.6	Big Learning Rate	13
2.7	Motion of a Deformable Body	15
2.8	Plane Stress Problem - (G. R. Liu, 2003)	18
2.9	Plane Strain Problem - (G. R. Liu, 2003)	19
2.10	Continuous solid subjected to volume forces and external forces - (Belinha, 2014)	20
2.11	Finite element discretisation of the 2D domain - (Finite Element Method course notes)	22
2.12	4-node quadrilateral element - (Madhujit Mukhopadhyay, 2022)	24
3.1	Neural Network performance compared with the FEM, for a particular bone geometry - Pais et al. (2023b)	29
3.2	Mean Absolute Error (MAE) at four different levels of fetal descent - Moura et al. (2024)	30
3.3	Stress Fields - Ribeiro et al. (2021a)	31
3.4	Performance of the ML models - Badarinath et al. (2021)	32
4.1	Plate with central hole	34
4.2	Plate Boundary Conditions	37
4.3	.csv file containing the node displacements	38
4.4	Sensitivity analysis of hyperparameters for the ANN model - Without Outliers	42
4.5	Sensitivity analysis of hyperparameters for the ANN model - With Outliers	43
4.6	Validation MAE across Epochs	44
4.7	Different model predictions of the displacement in x direction	44
4.8	Hyperparameter sensitivity analysis for the ANN model - Without Outliers	46
4.9	Hyperparameter sensitivity analysis for the ANN model - With Outliers	47
4.10	Best Validation MAE across Epochs	49
4.11	Comparison of FEM and ANN solutions for displacement v - Without Outliers	50
4.12	Comparison of FEM and ANN solutions for displacement v - With Outliers	51
4.13	Hyperparameter sensitivity analysis for the ANN model - Excluding Outliers	52
4.14	Hyperparameter sensitivity analysis for the ANN model - With Outliers	53
4.15	Validation MAE across Epochs	54
4.16	Comparison of FEM and ANN solutions for the stress - With and Without Outliers	55
4.17	Hyperparameter sensitivity analysis for the ANN model - No Outliers	56

4.18	Hyperparameter sensitivity analysis for the ANN model - With Outliers	57
4.19	Validation MAE across Epochs	58
4.20	Comparison of FEM and ANN solutions for the stress σ_{yy} - With and Without Outliers	59
4.21	Hyperparameter sensitivity analysis for the ANN model - Without Outliers	60
4.22	Hyperparameter sensitivity analysis for the ANN model - With Outliers	61
4.23	Validation MAE across Epochs	62
4.24	Comparison of FEM and ANN for the shear stress τ_{xy} - With and Without Outliers	63
4.25	Hyperparameter sensitivity analysis for the ANN model - Without Outliers	64
4.26	Hyperparameter sensitivity analysis for the ANN model - With Outliers	65
4.27	Validation MAE across Epochs	66
4.28	Comparison of FEM and ANN for the strain ϵ_{xx} - With and Without Outliers	67
4.29	Hyperparameter sensitivity analysis for the ANN model - Without Outliers	68
4.30	Hyperparameter sensitivity analysis for the ANN model - With Outliers	69
4.31	Validation MAE across Epochs	69
4.32	Comparison of FEM and ANN for the strain ϵ_{yy} - With and Without Outliers	71
4.33	Hyperparameter sensitivity analysis for the ANN model - Without Outliers	72
4.34	Hyperparameter sensitivity analysis for the ANN model - With Outliers	73
4.35	Validation MAE across Epochs	74
4.36	Comparison of FEM and ANN for the shear strain ϵ_{xy} - With and Without Outliers	75
4.37	Predictions at the nodes for stress σ_{xx}	76
4.38	Predictions at the nodes for stress σ_{yy}	77
4.39	Predictions at the nodes for stress τ_{xy}	78
4.40	Predictions at the nodes for strain ϵ_{xx}	79
4.41	Predictions at the nodes for strain ϵ_{yy}	80
4.42	Predictions at the nodes for strain ϵ_{xy}	81
5.1	Plate with four holes	84
5.2	Plate Boundary Conditions	86
5.3	Validation MAE along 1000 epochs - u displacement	87
5.4	Comparison of FEM and ANN predictions for displacement u across three different plate dimensions	89
5.5	Validation MAE along 1000 epochs - v displacement	90
5.6	Comparison of FEM and ANN predictions for displacement v across three different plate dimensions	92
5.7	Validation MAE along 2000 epochs - Stress σ_{xx}	93
5.8	Comparison of FEM and ANN predictions for Stress σ_{xx} across three different plate dimensions	95
5.9	Validation MAE along 2000 epochs - Stress σ_{yy}	96
5.10	Comparison of FEM and ANN predictions for Stress σ_{yy} across three different plate dimensions	97
5.11	Validation MAE along 2000 epochs - Stress τ_{xy}	98
5.12	Comparison of FEM and ANN predictions for Shear Stress τ_{xy} across three different plate dimensions	99
5.13	Validation MAE across the epochs - Strain ϵ_{xx}	100
5.14	Comparison of FEM and ANN predictions for Strain ϵ_{xx} across three different plate dimensions	102
5.15	Validation MAE across the epochs - Strain ϵ_{yy}	103

5.16 Comparison of FEM and ANN predictions for Strain ε_{yy} across three different plate dimensions	104
5.17 Validation MAE across the epochs - Strain ε_{xy}	105
5.18 Comparison of FEM and ANN predictions for Strain ε_{xy} across three different plate dimensions	106
5.19 Predictions at the nodes for stress σ_{xx}	107
5.20 Predictions at the nodes for stress σ_{yy}	108
5.21 Predictions at the nodes for stress τ_{xy}	109
5.22 Predictions at the nodes for strain ε_{xx}	110
5.23 Predictions at the nodes for strain ε_{yy}	111
5.24 Predictions at the nodes for strain ε_{xy}	112

Notation

AAA	Abdominal Aortic Aneurysms
AdaGrad	Adaptive Gradient Descent
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
cGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Networks
CPS4	4 Node Plane Stress Element
CSV	Comma Separated Value
DL	Deep Learning
FEM	Finite Element Method
FEA	Finite Element Analysis
K_t	Stress Concentration Factor
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MLPs	Multilayer Perceptrons
MRE	Mean Relative Error
MSE	Mean Squared Error
PFM	Pelvic Floor Muscles
PDEs	Partial Differential Equations
R	Correlation Coefficient
R^2	Determination Coefficient
RF	Random Forest
ReLU	Rectified Linear Unit
RMSProp	Root Mean Square Propagation
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
SCSNet	Single Input Channel Network
SGD	Stochastic Gradient Descent
SPR	Superconvergent Patch Recovery
SSIM	Structural Similarity Index Scores
StressNet	Multichannel Input Network
SVR	Support Vector Regression
XGBT	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Framework and Motivation

The Finite Element Method (FEM) is a widely used numerical method in engineering, extensively employed to solve partial differential equations. The origins of the FEM can be traced back to the early 1940s, with its formal development attributed to engineers and mathematicians who sought to solve complex elasticity and structural analysis problems. Richard Courant is credited with laying the groundwork for FEM in 1943 through his work on variational calculus. However, it wasn't until the 1950s and 1960s that FEM gained significant traction, thanks to the contributions of pioneers like J.H. Argyris, R.W. Clough, O.C. Zienkiewicz, and Ray W. Clough, who coined the term "finite element" in 1960. Since then, FEM has evolved into a fundamental tool in computational mechanics, offering powerful solutions for a wide range of engineering problems. (Liu et al., 2022)

The FEM works by subdividing a large and complex domain into smaller and more manageable subdomains known as finite elements through a process called space discretization. The finite elements form a mesh, where the governing equations are approximated and solved over each element, leading to a solution for the entire domain. This method is particularly advantageous for dealing with complex geometries, material properties, and boundary conditions that are otherwise difficult to solve analytically.

As the complexity of numerical applications has grown, traditional methods such as the FEM encountered challenges. These methods demand substantial computational resources and incur high computational costs, especially when dealing with complex or large scale problems. In the 21st century, even with advancements in computational power, the time required to perform such analyses remains a critical bottleneck.

To address these challenges, the integration of Artificial Intelligence (AI) techniques, particularly Machine Learning (ML), as emerged as a promising solution. Machine Learning (ML), a subset of AI, has been increasingly combined with the FEM in order to reduce the computational costs associated with the numerical analysis. The ML techniques offer the capability to make predictions while significantly decreasing computational time. In recent years, there has been a

growth in the number of studies that successfully leverage ML techniques within the context of Finite Element Analysis (FEA), often serving as surrogate solvers or predictive tools.

This dissertation focuses on the development of a Machine Learning approach, specifically an Artificial Neural Network (ANN), to serve as a surrogate for standard Finite Element Analysis in linear elasticity. The proposed method involves constructing the ANN, which will be used to predict the structural response of well known Solid Mechanics benchmarks.

The study begins by analysing the problem of a two dimensional plate with a central hole subjected to in plane loads. Following this, the research extends to examine a plate with four identical holes, also subjected to a distributed load. These cases are chosen for their relevance in practical engineering applications and their use as benchmarks in the field.

Preliminary results and existing studies indicate that Artificial Neural Networks are capable of accurately predicting displacements, stresses and strains while significantly reducing the overall computational time. The literature suggests that, on average, computational time can be reduced by as much as 90%, leading to a significant reduction in costs. However, a notable challenge of ANNs is the need to acquire substantial amounts of data and adequately prepare it for training and testing, which can be time consuming at a first stage.

1.2 Objectives

The principal and main objectives of this thesis are the following:

- Develop an Artificial Neural Network (ANN) model to predict the structural response (displacements, stresses and strains) in linear elasticity problems
- Integrate the developed ANN model with FEM to serve as a surrogate solver
- Train and validate the ANN to well known solid mechanics benchmarks, to assess its predictive accuracy
- Compare the FEM solutions with the ANN predictions
- Explore the influence of data quality, particularly the influence of outliers on the performance of the ANN model
- Analyse whether the neural network trained on the integration points can make predictions for the nodes, assessing the model's versatility

1.3 Dissertation Structure

Besides introduction, this dissertation will contain five more chapters.

In Chapter 2, the theoretical background is presented, covering essential concepts related to Artificial Neural Networks and the Finite Element Method, providing the foundation for the study.

Chapter 3 reviews the state of the art, which encompasses relevant studies that integrate Artificial Neural Networks with the Finite Element Method, highlighting the advancements and challenges in this field.

Chapter 4 focuses on the problem of a plate with a circular hole. The chapter details the problem, outlines the methodology employed, and discusses the model's results in predicting each output, including an analysis of the findings.

Chapter 5 encompasses the case of the plate with four equal circular holes. The problem is described and the methodology is outlined. The results of the predictions for each output using the optimal hyperparameters of the first case are presented and discussed.

Finally, Chapter 6 presents the study conclusions, outlines its limitations and offers recommendations for future work.

Chapter 2

Theoretical Background

In this chapter the focus will be on presenting the essential topics to the forthcoming work. Initially, a concise overview of Artificial Intelligence, with a specific focus on Artificial Neural Networks as a subset of Deep Learning, will be provided. Following this, the fundamental principles of Solid Mechanics will be presented in order to establish the groundwork for understanding the Finite Element Method.

2.1 Artificial Intelligence

Artificial Intelligence (or AI) is increasingly starting to have a normal presence in our daily lives. It offers the solution for a wide range of problems and the advantages it has brought to society in recent years has demarcated its importance. It isn't anymore a futuristic concept since it is already transforming industries and boosting human capabilities.

AI is a broad term that covers the development of machines capable of imitating human behavior. In theory, hard coding a task could qualify as Artificial Intelligence if all potential scenarios are pre-programmed into the system. However, the prevailing approach in AI today is soft coding, where the system is designed to learn and adapt with minimal human participation. (Pais et al., 2023b)

Within AI soft coding, Machine Learning (ML) and Deep Learning (DL), both considered subsets of AI, enable systems to learn and enhance their performance through experience.

2.1.1 Machine Learning

Machine Learning (ML) is a sub field of AI that enables computers to learn from data and experience without being explicitly programmed. ML is a field of data science that employs computer algorithms and analytical techniques to construct models capable of making classifications and predictions. (Pais et al., 2023a)

Machine Learning algorithms are generally divided into two main types: Supervised and Un-supervised Learning. Supervised learning is defined by the use of labeled datasets to train algorithms to classify data or predict outcomes accurately. Essentially, these algorithms are designed

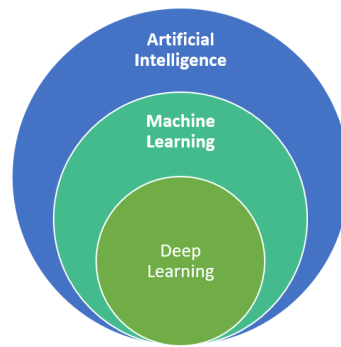


Figure 2.1: Subsets of AI

to learn associations between input data x and their corresponding outputs y based on a provided training set.

Unsupervised learning uses ML algorithms to analyse and cluster unlabeled datasets. In this approach, the algorithms are tasked with autonomously discerning patterns, structures, or relationships within the data, without relying on predefined target labels.

2.1.2 Deep Learning

Deep Learning (DL) is a branch of AI, and more precisely, a subfield of Machine Learning (ML). It powers numerous AI driven applications and services, enhancing automation by carrying out both analytical and physical tasks without the need for human involvement.

Deep Learning distinguishes itself from classical Machine Learning by the type of data that it works with and the methods in which it learns. Machine Learning algorithms use organized and labeled data to make predictions. This involves defining specific features from the input data and arranging them in tables. While Machine Learning can also handle unstructured data, it usually requires some pre-processing to organise it into a structured format before making predictions. (IBM, 2017)

Deep Learning eliminates the data pre-processing that is typically involved with Machine Learning. These algorithms can absorb and process unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts. Deep Learning algorithms excel at identifying features for discerning between various entities in contrast with traditional Machine Learning. (IBM, 2017)

2.2 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) are the backbone of Deep Learning algorithms. They are called "neural" because they are inspired by the human neurological system. Perhaps the first ANN architectures were made to simulate the human brain, they have become powerful tools to solve a great variety of problems. One of their most common applications is pattern recognition, where the network is provided with input and output pairs and tasked with learning a function that accurately approximates the underlying relationships between them. (Ribeiro et al., 2021a)

Artificial Neural Networks (ANN) consist of layers of nodes, including an input layer, one or more hidden layers and an output layer. Every node, or neuron, establishes connections with others and possesses a weight and threshold. Activation occurs when the output of an individual node surpasses the specified threshold, leading to the transmission of data to the subsequent layer of the network. Conversely, if the output falls below the threshold, no data is forwarded to the next layer. (López et al., 2022)

In addition to Artificial Neural Networks (ANNs), there are other types of neural networks, such as Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

ANN (Figure 2.2a) are often referred to as Feed Forward Neural Networks, as information flows in a single direction, from input to output.

RNN (Figure 2.2b) are designed to capture the sequential patterns within the input data while making predictions. This allows for fewer parameters to be trained, leading to reduced computational costs.

Lastly, CNNs (Figure 2.2c) automatically learn filters that help in extracting relevant features from the input data. They are commonly applied to image data but can also be used with sequential data. (Pais et al., 2023a)

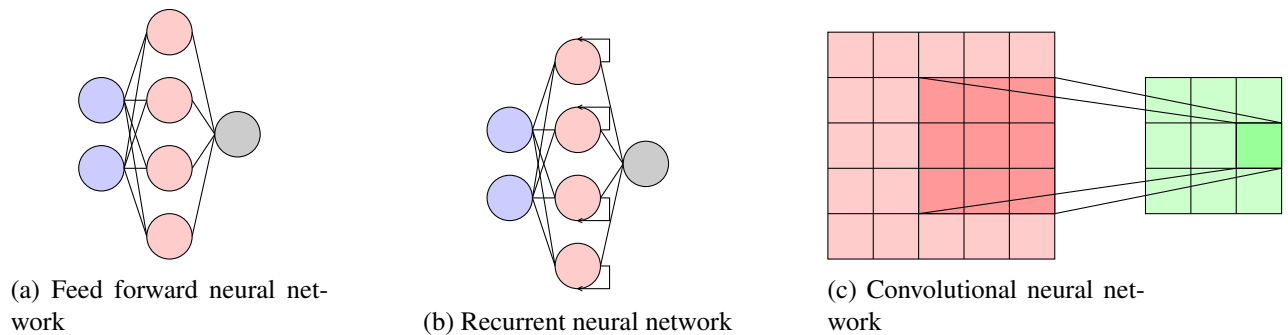


Figure 2.2: Types of artificial neural networks

Feed Forward Neural Networks, also known as Multilayer Perceptrons (MLPs) for having multiple layers of neurons, represent a fundamental class of Deep Learning models. One example of it is shown in Figure 2.3.

The basic unit of a Multilayer Perceptron is a neuron, often named as perceptron, and its function is quite simple to understand. A weighted sum of signals arrives at the input node and is

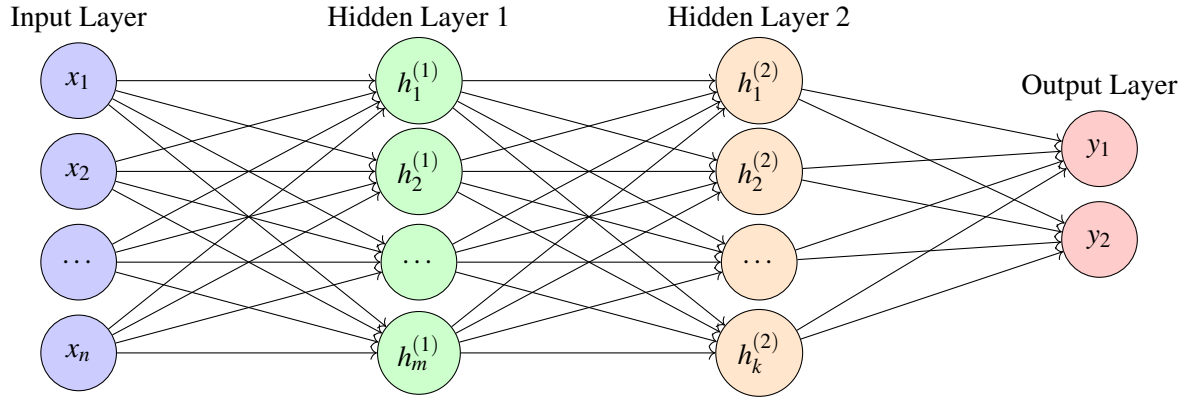


Figure 2.3: Example of an ANN

then subjected to a transformation function f . Equation 2.1 shows the nonlinear transformation f applied to the bias and the weighted sum of input values.

$$z = f(b + xw) = f\left(b + \sum_{i=1}^m x_i w_i\right) \quad (2.1)$$

where b is the bias, x_i is the value from the node i in the previous layer and w_i is the weight associated with the same node. (Pais et al., 2023b)

The activation function determines the output of a node based on its inputs and their corresponding weights. Activation functions can be categorized into two types:

1. Linear Activation Functions
2. Non-Linear Activation Functions

Below are examples of commonly used activation functions. The Sigmoid activation function, which is non-linear, is defined as:

$$f(\Sigma) = \frac{1}{1 + e^{-\Sigma}} \quad (2.2)$$

where Σ represents the weighted sum of inputs (Kubat, 2017).

The hyperbolic tangent activation function, or $\tanh$, is another non-linear function similar to the sigmoid function. However, the range of the $\tanh$ function is from -1 to 1. The $\tanh$ activation function is given by:

$$f(\Sigma) = \frac{e^{\Sigma} - e^{-\Sigma}}{e^{\Sigma} + e^{-\Sigma}} \quad (2.3)$$

The Rectified Linear Unit (ReLU) activation function is one of the most used activation functions. It outputs zero for negative inputs and returns the input value for non-negative inputs. The *ReLU* function is defined as:

$$f\left(\sum\right) = \max(0, x) \quad (2.4)$$

These activation functions play crucial roles in the performance and training of neural networks (Wang et al., 2020). They can be represented as,

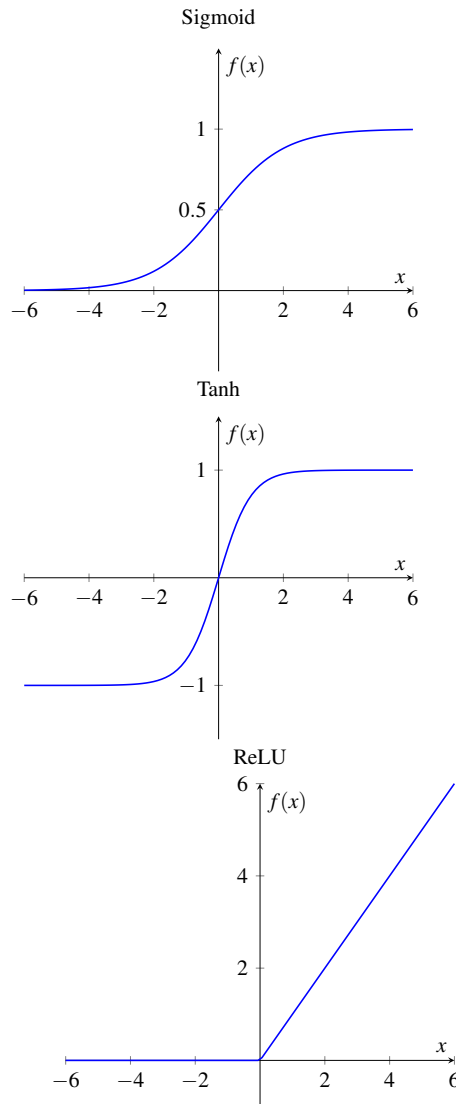


Figure 2.4: Activation Functions: Sigmoid, Tanh and ReLU

The accuracy or performance of neural network predictions can be evaluated using several metrics, with one of the most common being the Mean Squared Error (MSE). The MSE is calculated as the squared difference between the target vector and the output vector,

$$MSE = \frac{1}{m} \sum_{i=1}^m (t_i - y_i)^2 \quad (2.5)$$

In this formula, t_i corresponds to the target vector and y_i to the output vector at node i . Squaring the differences $(t_i - y_i)$ ensures that negative differences do not cancel out positive ones. (Kubat, 2017) However, a drawback of the MSE loss function is its sensitivity to outliers; predictions significantly deviating from their target values can disproportionately increase the loss.

Other accuracy metrics include the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE). (Pais et al., 2023a) The MAE is defined as the average of the absolute differences between the target vector t_i and the output vector y_i ,

$$MAE = \frac{1}{m} \sum_{i=1}^m |t_i - y_i| \quad (2.6)$$

Unlike the MSE, the MAE does not square the differences, making it less sensitive to outliers and providing a different perspective on prediction errors.

An optimization problem always seeks to minimize the loss function because reducing the error enhances the model's predictions. Minimizing the loss function helps in finding the optimal set of weights and biases, which in turn improves the model's performance and accuracy. (López et al., 2022)

Neural network training consists of two main phases: the forward propagation and backward propagation. During forward propagation, the network generates an output or prediction based on a given set of input data. The input data values are passed along the links to the neurons, where each value is multiplied by the weights associated with the respective links. The next neuron receives as input a weighted sum and, subjects it to an activation function f . (Kubat, 2017)

Backward propagation, on the other hand, is used to train the network by determining the error and adjusting the weights and biases. This process involves minimizing the loss function, which can be the Mean Square Error (MSE) or any other function. The difference between the output generated in forward propagation and the target vector informs the required adjustments needed for the weights. Each complete cycle of forward and backward propagation through the network is referred to as an *epoch*. (Pais et al., 2023a)

2.2.1 Optimization Algorithms

To effectively adjust weights and biases during training, optimization algorithms, commonly known as optimizers, are used. Optimizers play a crucial role in the training process by defining how the model's parameters are updated based on the gradients computed during back propagation. The primary objective of an optimizer is to minimize the loss function, thereby improving model performance. Consequently, the choice of optimizer can significantly impact the convergence speed and quality of the model.

Several optimizers are commonly discussed, including Gradient Descent, Stochastic Gradient Descent (SGD), Adaptive Gradient Descent (AdaGrad), Root Mean Square Propagation (RMSProp) and Adaptive Moment Estimation (Adam).

1. Gradient Descent

The Gradient Descent is the most popular and basic optimization algorithm. This optimization algorithm updates the weights in the direction of the negative gradient of the loss function with respect to the weights, aiming to reach a global minimum. The weight update rule is given by:

$$w_{t+1} = w_t - \alpha \nabla L(w_t) \quad (2.7)$$

where w_t represents the weights at the time step t , α the learning rate and the $\nabla L(w_t)$ is the gradient of the loss function at w_t . (Goodfellow et al., 2017)

2. Stochastic Gradient Descent (SGD)

The Stochastic Gradient Descent addresses the inefficiency of the Gradient Descent when dealing with large datasets. Instead of processing the entire dataset for each iteration, SGD randomly (hence the term Stochastic) selects batches of data and updates the model parameters. This approach is more computationally efficient and leads to faster convergence but can result in noisy updates.

3. AdaGrad (Adaptive Gradient Descent)

AdaGrad is a variant of the gradient descent algorithm that adapts the learning rate for each parameter individually. It applies smaller updates to parameters linked to frequently occurring features, while making larger updates for those associated with less common features. With this, the *AdaGrad* has the ability to handle sparse data. The weight update rule is given by,

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla L(w_t) \quad (2.8)$$

where G_t is the sum of the squares of the past gradients for each parameter, and ϵ is a small constant to prevent division by zero. (Goodfellow et al., 2017)

4. RMSProp (Root Mean Square Propagation)

The RMSProp was designed to resolve the diminishing learning rates issue encountered with AdaGrad. It adjusts the learning rate for each parameter by dividing the gradient by a moving average of the magnitudes of recent gradients. The weight update rule is given by,

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)g_t^2 \quad (2.9)$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}} \nabla L(w_t) \quad (2.10)$$

Where $E[g^2]_t$ is the moving average of the squared gradients, β is a decay rate and ε is a constant. The RMSProp helps maintain a stable learning rate and converges quickly. (Goodfellow et al., 2017)

5. Adam (Adaptive Moment Estimation)

The Adam optimizer combines the advantages of both AdaGrad and RMSProp, making it well suited for a wide range of machine learning problems.

- **Learning Rate:** Adam adjusts the learning rate for each parameter individually by computing running averages of the gradient and its squared value. This means that parameters with larger gradients will have smaller learning rates, while parameters with smaller gradients will have larger learning rates.
- **Momentum:** The optimizer incorporates momentum, which helps accelerate gradient vectors in the right direction, leading to faster convergence. It computes the exponentially decaying average of past gradients, which helps smooth the gradient descent path

The mathematical formulation for the Adam optimizer consists on the following steps. Firstly, initialize the first moment vector m and second moment vector v to zero, as well as the time step t .

$$m_0 = 0, \quad v_0 = 0, \quad t = 0 \quad (2.11)$$

At each time step t , compute the gradient of the loss function with respect to the current weights w_t ,

$$g_t = \nabla L_t(w_t) \quad (2.12)$$

Then, calculate the exponentially decaying average of past gradients (first moment estimate),

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t \quad (2.13)$$

Next, calculate the exponentially decaying average of past squared gradients (second raw moment estimate),

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.14)$$

Correct the bias in the first moment estimate and second moment estimate,

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (2.15)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.16)$$

Finally, update the parameters using the computed bias-corrected estimates.

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (2.17)$$

where α is the learning rate, β_1 is the decay rate for the first moment estimate, β_2 the decay rate for the second moment estimate and ϵ the constant to prevent the division from 0. (Goodfellow et al., 2017)

The Adam optimizer is a powerful tool for training neural networks, as it effectively adapts learning rates and incorporates momentum. This adaptability and momentum enable efficient and effective optimization, significantly enhancing model performance and convergence.

2.2.2 Learning Rate

The learning rate, denoted as α , is a crucial parameter in the optimization process of neural networks. It determines the size of the steps taken towards minimizing the loss function. A high learning rate results in larger steps, which can speed up the training process but also risks overshooting the optimal minimum. On the other hand, a low learning rate results in smaller, more precise steps, which can lead to better convergence but may require more iterations and thus more time and computational resources.

Finding the right balance for the learning rate is essential. If the learning rate is too high, the model may become unstable and fail to converge. If it is too low, the training process may be excessively slow, potentially leading to getting stuck in local minima. Therefore, choosing an appropriate learning rate is a key aspect of training neural networks efficiently and effectively. Techniques such as learning rate schedules and adaptive learning rates, as used in optimizers like Adam, can help dynamically adjust the learning rate during training to improve convergence and performance.

As illustrated in Figure 2.5, a smaller learning rate results in gradual and precise steps toward the minimum of the loss function. In contrast, Figure 2.6 shows that a larger learning rate leads to bigger, more erratic steps. These larger steps can cause the optimization process to overshoot the minimum, leading to instability in the training process.

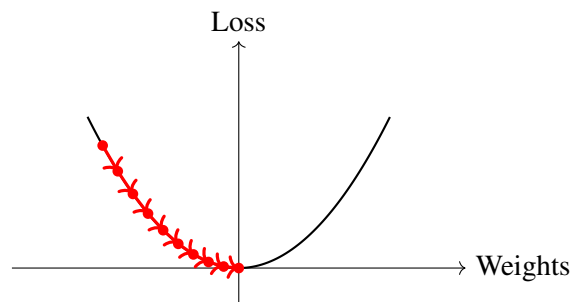


Figure 2.5: Small Learning Rate

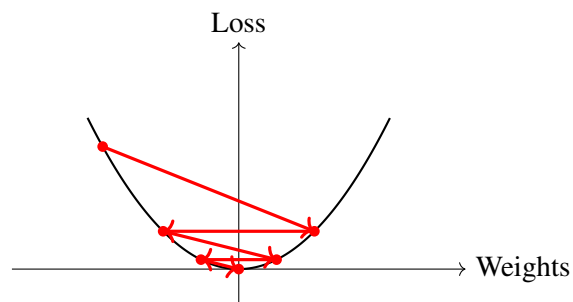


Figure 2.6: Big Learning Rate

2.3 Solid Mechanics Fundamentals

Solid Mechanics stands as a cornerstone within physical science, focusing on the intricate interplay between external loading and the resulting deformation and motion of continuous solid materials. It delves into the complexities of how structures respond to applied forces, displacements, induced stresses, and so on. By knowing the configuration of the structure as well as the loads applied it is possible to obtain the strain and stress fields applied using known relations. (Sancaktar, 2008)

This work exclusively focus on linear elastic materials, where the deformation disappears entirely if the solid is unloaded. Furthermore, given that this is a static study, the stress, strain and displacements will not be functions of time. In addition, this work addresses only isotropic materials, where the material properties are not direction dependent. (G. R. Liu, 2003)

This section delves into the fundamental principles of Solid Mechanics, laying the groundwork necessary for a thorough comprehension of the subsequent chapters.

2.3.1 Stress

Consider a 3D solid with a certain volume V and a surface Γ . The solid is subjected to external forces and, as a result, internal forces are produced. These internal forces are called stresses, which are defined by the amount of force over a certain area.

At any point within the solid, the stress components can be conceptualized by considering an infinitesimally small cubic volume. On each surface, there are three stress components: a normal stress component σ and two components of shearing stress τ . The subscript notation follows a sign convention where the first letter denotes the surface on which the stress acts, and the second letter indicates the direction of the stress. (Timoshenko and Goodier, 1951)

The stress experienced by a body at a specific point is described by the stress tensor and it can be defined as,

$$\sigma = \begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{bmatrix} \quad (2.18)$$

The Voigt notation is employed for the sake of simplification. As so, the stress tensor can be written in vector form as,

$$\sigma = \{\sigma_{xx} \quad \sigma_{yy} \quad \sigma_{zz} \quad \tau_{xy} \quad \tau_{yz} \quad \tau_{xz}\}^T \quad (2.19)$$

2.3.2 Strain

When external forces act upon a body, deformation occurs as a natural response. Since no body is perfectly rigid, any displacement of its particles necessitates a corresponding deformation.

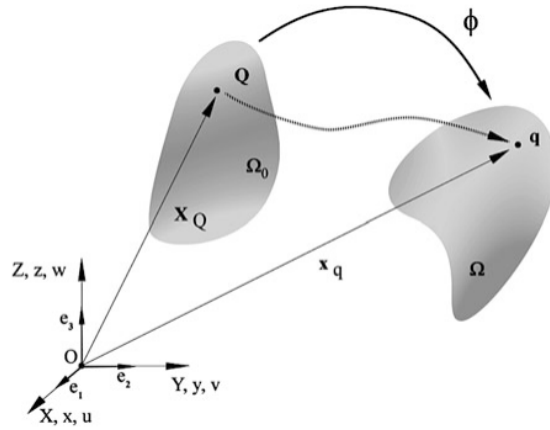


Figure 2.7: Motion of a Deformable Body

Referring to Figure 2.7 where a point $\mathbf{Q}$ changes its position to $\mathbf{q}$ when subjected to an external load. For any given point of the solid, the equation that defines the displacement is expressed as follows,

$$\mathbf{u}(u, v, w) = \begin{Bmatrix} u(x, y, z) \\ v(x, y, z) \\ w(x, y, z) \end{Bmatrix} \quad (2.20)$$

Strain is the change of displacement per unit length therefore, the components of strain can be obtained from the derivatives of the displacements,

$$\epsilon_{xx} = \frac{\partial u}{\partial x}; \quad \epsilon_{yy} = \frac{\partial v}{\partial y}; \quad \epsilon_{zz} = \frac{\partial w}{\partial z}; \quad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}; \quad \gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}; \quad \gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \quad (2.21)$$

where u, v and w are the displacement components. The normal strain is represented by ϵ and the shear strain represented by γ . (G. R. Liu, 2003)

The strain tensor is reduced to the strain vector and is represented as,

$$\boldsymbol{\epsilon} = \{\epsilon_{xx} \quad \epsilon_{yy} \quad \epsilon_{zz} \quad \gamma_{xy} \quad \gamma_{xz} \quad \gamma_{yz}\}^T \quad (2.22)$$

The strain-displacements relations in Equation (2.21) can be rewritten in the matrix form,

$$\boldsymbol{\epsilon} = \mathbf{L}\mathbf{u} \quad (2.23)$$

where $\mathbf{u}$ is the displacement vector and $\mathbf{L}$ is the matrix of partial differential operator. $\mathbf{L}$ is given by,

$$\mathbf{L} = \begin{bmatrix} \partial/\partial x & 0 & 0 \\ 0 & \partial/\partial y & 0 \\ 0 & 0 & \partial/\partial z \\ \partial/\partial y & \partial/\partial x & 0 \\ \partial/\partial z & 0 & \partial/\partial x \\ 0 & \partial/\partial z & \partial/\partial y \end{bmatrix} \quad (2.24)$$

2.3.3 Constitutive Equations

The constitutive equation gives the relationship between the stress and the strain of the solid. It is known as the Hooke's Law. The generalised Hooke's Law is given by,

$$\boldsymbol{\sigma} = \mathbf{c} \boldsymbol{\varepsilon} \quad (2.25)$$

where $\mathbf{c}$ represents the material constitutive matrix. As previously stated, all solids in this work are considered linear elastic and isotropic. Consequently, the matrix $\mathbf{c}$ is defined by,

$$\mathbf{c} = \begin{bmatrix} c_{11} & c_{12} & c_{12} & 0 & 0 & 0 \\ & c_{11} & c_{12} & 0 & 0 & 0 \\ & & c_{11} & 0 & 0 & 0 \\ & & & (c_{11} - c_{12})/2 & 0 & 0 \\ sym. & & & & (c_{11} - c_{12})/2 & 0 \\ & & & & & (c_{11} - c_{12})/2 \end{bmatrix} \quad (2.26)$$

where

$$c_{11} = \frac{E(1-\nu)}{(1-2\nu)(1+\nu)}; \quad c_{12} = \frac{E\nu}{(1-2\nu)(1+\nu)}; \quad \frac{c_{11} - c_{12}}{2} = G \quad (2.27)$$

in which E is the Young's Modulus, ν the Poisson's ratio and G the shear modulus of the material. The relation of these constants is given by,

$$G = \frac{E}{2(1+\nu)} \quad (2.28)$$

For any isotropic material, if two of the three constants are known, the third one can be determined using the equation provided above. (G. R. Liu, 2003)

2.3.4 Equilibrium Equations

The equilibrium equations describe the relationship between the stress and external forces that are being applied on the solid. The equilibrium equations are represented by a system of partial differential that need to be satisfied for every x inside the domain:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + b_x = 0 \quad (2.29)$$

$$\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + b_y = 0 \quad (2.30)$$

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + b_z = 0 \quad (2.31)$$

Written in a more compact form as,

$$\nabla \sigma + \mathbf{b} = 0 \quad (2.32)$$

where $\mathbf{b}$ is the body force vector,

$$\mathbf{b} = \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} \quad (2.33)$$

2.4 Equations for two-dimensional solids

The equations outlined above are formulated for three-dimensional solids. However, it is possible to simplify and analyse three dimensional problems as two dimensional solids. For the representation of a 2D solid the coordinate z is removed, assuming that all dependent variables are independent of the z – axis and that external loads act only in the x and y planes. There are two types of 2D solids, one being the *plane stress* solid and other the *plane strain* solid. (G. R. Liu, 2003)

2.4.1 Plane Stress

For the plane stress solids the x and y dimensions are much larger than the thickness z . External forces are applied only in the $x - y$ plane and the normal stress σ_{zz} and the shear stresses τ_{xz} and τ_{yz} are zero, as shown in Figure 2.8.

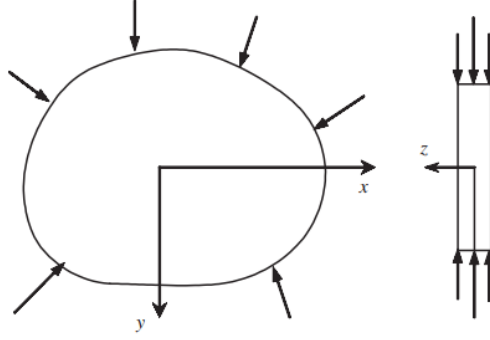


Figure 2.8: Plane Stress Problem - (G. R. Liu, 2003)

The stress and strain tensor can be represented in vector form as follows,

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} \quad (2.34)$$

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix} \quad (2.35)$$

The material matrix $\mathbf{c}$, for the isotropic material, is given by

$$\mathbf{c} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \quad (2.36)$$

2.4.2 Plane Strain

Plane strain solids are solids whose thickness z is much larger when compared with the other dimensions. External forces perpendicular with its thickness are applied while maintaining the same cross-section, as shown in Figure 2.9. The normal stress σ_{zz} and the shear stresses τ_{xz} and τ_{yz} can also be considered zero.

The stress and strain vectors take the same form as Equations (2.34) and (2.35). The material matrix $\mathbf{c}$ takes the form,

$$\mathbf{c} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\nu/2 \end{bmatrix} \quad (2.37)$$

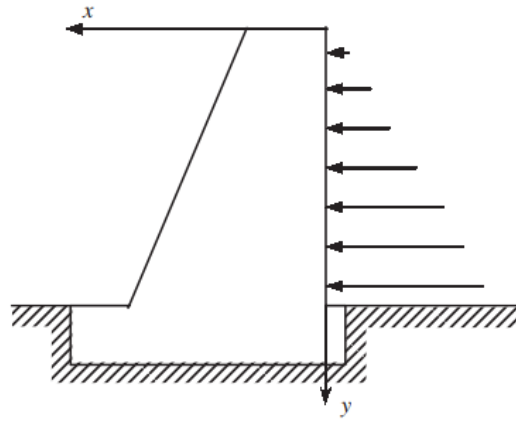


Figure 2.9: Plane Strain Problem - (G. R. Liu, 2003)

2.5 Galerkin Weak Form

An equilibrium problem can be expressed in Strong and Weak Forms. The Strong Form consists on a system of partial differential equations in space or time, associated with appropriate boundary conditions to be verified locally, whereas the Weak Form emerges as a weighted integral form, which replaces local equilibrium verification in all the points of the domain into an average equilibrium condition. (G. R. Liu, 2003)

Obtaining an exact solution from the Strong Form system is often challenging, particularly for complex engineering problems. On the contrary, Weak Form based formulations can generate stable algebraic equations and provide a discretized system that leads to more accurate solutions. All of these factors contribute to the preference of the Weak Form to obtain the approximated solution. (Belinha, 2014)

The Galerkin Weak Form is a variational principle based on an energy principle. Hamilton's principle, which is one of the most fundamental energy principles, states that of all possible displacement configurations satisfying the compatibility equations, the essential or the kinematic boundary conditions and the initial and final time conditions, the real solution correspondent configuration is the one that minimizes the Lagrangian functional L ,

$$L = T - U + W_f \quad (2.38)$$

being T the kinetic energy, U the potential energy (which for our purpose, is the strain energy) and W_f is the work produced by external forces.

Considering a solid body with a domain denoted by Ω bounded by Γ , where external forces $\bar{t}$ are applied on the surface defined by the natural boundary condition Γ_t , displacements are constrained on the surface Γ_u and a body force b applied in the continuous solid, as represented in Figure 2.10.

The kinetic energy is defined as,

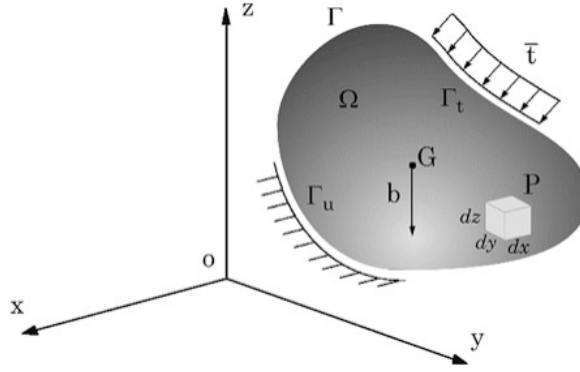


Figure 2.10: Continuous solid subjected to volume forces and external forces - (Belinha, 2014)

$$T = \frac{1}{2} \int_{\Omega} \rho \dot{\mathbf{u}}^T \dot{\mathbf{u}} d\Omega \quad (2.39)$$

where $\dot{\mathbf{u}}$ represents the velocity, displacement first derivative, and ρ the solid mass density. The strain energy, for linear elastic materials, is defined by,

$$U = \frac{1}{2} \int_{\Omega} \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega \quad (2.40)$$

being $\boldsymbol{\varepsilon}$ the strain vector and $\boldsymbol{\sigma}$ the stress vector. The work done by the external forces can be expressed as,

$$W_f = \int_{\Omega} \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \quad (2.41)$$

where $\mathbf{u}$ represents the displacement. By substituting the Equations 2.39, 2.40 and 2.41 in the Lagrangian functional L it gives,

$$L = \frac{1}{2} \int_{\Omega} \rho \dot{\mathbf{u}}^T \dot{\mathbf{u}} d\Omega - \frac{1}{2} \int_{\Omega} \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega + \int_{\Omega} \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \quad (2.42)$$

and then minimized,

$$\delta \int_{t_1}^{t_2} \left[\frac{1}{2} \int_{\Omega} \rho \dot{\mathbf{u}}^T \dot{\mathbf{u}} d\Omega - \frac{1}{2} \int_{\Omega} \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega + \int_{\Omega} \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \right] dt = 0 \quad (2.43)$$

Moving the variation operator δ inside the integrals,

$$\int_{t_1}^{t_2} \left[\frac{1}{2} \int_{\Omega} \delta(\rho \dot{\mathbf{u}}^T \dot{\mathbf{u}}) d\Omega - \frac{1}{2} \int_{\Omega} \delta(\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}) d\Omega + \int_{\Omega} \delta \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \right] dt = 0 \quad (2.44)$$

Since only static problems were considered in this work, the Kinetic Energy T becomes zero. So, the first term of Equation (2.44) can be dismissed, which leads to,

$$\int_{t_1}^{t_2} \left[-\frac{1}{2} \int_{\Omega} \delta(\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}) d\Omega + \int_{\Omega} \delta \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \right] dt = 0 \quad (2.45)$$

The first term of Equation (2.45) can be developed. The integrand function can be written as,

$$\delta(\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}) = \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} + \boldsymbol{\varepsilon}^T \delta \boldsymbol{\sigma} \quad (2.46)$$

given that the two terms in Eq. (2.46) are scalars,

$$\boldsymbol{\varepsilon}^T \delta \boldsymbol{\sigma} = (\boldsymbol{\varepsilon}^T \delta \boldsymbol{\sigma})^T = \delta \boldsymbol{\sigma}^T \boldsymbol{\varepsilon} \quad (2.47)$$

Using the constitutive equation $\boldsymbol{\sigma} = \mathbf{c} \boldsymbol{\varepsilon}$, and the symmetric property of the material matrix, $\mathbf{c} = \mathbf{c}^T$, it is possible to write,

$$\delta \boldsymbol{\sigma}^T \boldsymbol{\varepsilon} = \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} \quad (2.48)$$

Substituting (2.48) in Equation (2.46),

$$\delta(\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}) = 2\delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} \quad (2.49)$$

and by replacing (2.49) in Equation (2.45),

$$\int_{t_1}^{t_2} \left[-\int_{\Omega} \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega + \int_{\Omega} \delta \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma \right] dt = 0 \quad (2.50)$$

In order to meet Equation (2.50) for all conceivable selections of the integrand within the time integration, it must be equal to zero (null). This condition results in the subsequent expression:

$$-\int_{\Omega} \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega + \int_{\Omega} \delta \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma = 0 \quad (2.51)$$

This last equation is known as the "Galerkin Weak Form" or Principle of Virtual Work. The Principle of Virtual Work states that a solid body is in equilibrium when the internal virtual work equals the external virtual work for every virtual displacement field. (Harvard, 2007) Considering the stress-strain relation, $\sigma = c\varepsilon$, and the strain-displacement relation, $\varepsilon = \mathbf{L}\mathbf{u}$, Equation (2.51) can be rearranged as,

$$\int_{\Omega} \delta(\mathbf{L}\mathbf{u})^T \mathbf{c}(\mathbf{L}\mathbf{u}) d\Omega - \int_{\Omega} \delta \mathbf{u}^T \mathbf{b} d\Omega - \int_{\Gamma_t} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma = 0 \quad (2.52)$$

This represents the generic Galerkin Weak Form in terms of displacements.

2.6 Finite Element Procedure

The FEM process begins with discretizing the domain, where a solid body is divided into N_e elements. This step, known as meshing, is typically carried out with the help of pre-processors, which assigns unique identifiers to all elements and nodes in the structure. Each element is formed connecting its nodes in a consistent, pre-defined way to create the connectivity of the element, forming the entire problem domain without any gap or overlapping. (G. R. Liu, 2003)

The second stage of the Finite Element Method requires defining the shape functions for each element. The shape functions describe the element's behaviour.

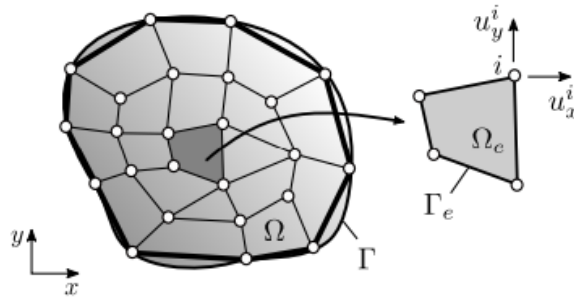


Figure 2.11: Finite element discretisation of the 2D domain - (Finite Element Method course notes)

Given a 2D element with the domain denoted by Ω_e and a boundary denoted by Γ_e , the element displacement vector is given by,

$$\mathbf{d}^e = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & \dots & u_n & v_n \end{bmatrix}^T \quad (2.53)$$

The displacement vector in each element is interpolated by the nodal displacements as

$$u = \sum_{i=1}^n N_i^e u_i; \quad v = \sum_{i=1}^n N_i^e v_i \quad (2.54)$$

where N_i^e represents the element shape functions. This can be expressed in matrix form as,

$$\mathbf{u} = \mathbf{N} \mathbf{d}^e = \begin{bmatrix} N_1^e & 0 & 0 & N_2^e & 0 & \dots & N_n^e & 0 \\ 0 & N_1^e & 0 & 0 & N_2^e & \dots & 0 & N_n^e \end{bmatrix} \mathbf{d}^e \quad (2.55)$$

The strain can be obtained by the derivation of the displacements. The strain field in the element is given by,

$$\boldsymbol{\varepsilon} = \mathbf{B} \mathbf{d}^e \quad (2.56)$$

where $\mathbf{B}$ is the strain displacement matrix defined by,

$$\mathbf{B} = \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \dots & \frac{\partial N_n^e}{\partial x} & 0 \\ 0 & \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial y} & \dots & 0 & \frac{\partial N_n^e}{\partial y} \\ \frac{\partial N_1^e}{\partial y} & \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial y} & \frac{\partial N_2^e}{\partial x} & \dots & \frac{\partial N_n^e}{\partial y} & \frac{\partial N_n^e}{\partial x} \end{bmatrix} \quad (2.57)$$

The discretised virtual work equation for the two dimensional element ($d\Omega = d\Omega^e = dx dy$) is given by,

$$\int_{\Omega^e} h^e \delta \mathbf{d}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{d}^e dx dy - \int_{\Omega^e} h^e \delta \mathbf{d}^{eT} \mathbf{N}^T \mathbf{b} dx dy - \int_{\Gamma_\sigma^e} h^e \delta \mathbf{d}^{eT} \mathbf{N}^T \bar{\mathbf{t}} d\Gamma_\sigma^e = 0 \quad (2.58)$$

where h^e corresponds to the thickness of the element. The finite element model equation is defined by,

$$\left[\int_{\Omega^e} h^e \mathbf{B}^T \mathbf{D} \mathbf{B} dx dy \right] \mathbf{d}^e = \int_{\Omega^e} h^e \mathbf{N}^T \mathbf{b} dx dy + \int_{\Gamma_\sigma^e} h^e \mathbf{N}^T \bar{\mathbf{t}} d\Gamma_\sigma^e \quad (2.59)$$

where the element stiffness matrix $\mathbf{K}^e$ is described as,

$$\mathbf{K}^e = \int_{\Omega^e} h^e \mathbf{B}^T \mathbf{D} \mathbf{B} dx dy \quad (2.60)$$

and the vector of external forces $\mathbf{f}^e$ corresponds to,

$$\mathbf{f}^e = \int_{\Omega^e} h^e \mathbf{N}^T \mathbf{b} dx dy + \int_{\Gamma_\sigma^e} h^e \mathbf{N}^T \bar{t} d\Gamma_\sigma^e \quad (2.61)$$

This can be written in a simpler manner as,

$$\mathbf{K}^e \mathbf{d}^e = \mathbf{f}^e \quad (2.62)$$

For the element shape function, it is considered a quadrilateral element as represented in Figure (2.12). The element is defined by four nodes in natural coordinates (ξ, η) .

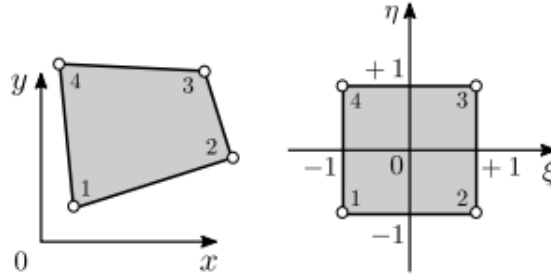


Figure 2.12: 4-node quadrilateral element - (Madhujit Mukhopadhyay, 2022)

The coordinates are interpolated as,

$$x = \sum_{i=1}^4 N_i^e x_i; \quad y = \sum_{i=1}^4 N_i^e y_i \quad (2.63)$$

where N_i are the Lagrange shape functions,

$$N_1(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 - \eta) \quad (2.64)$$

$$N_2(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 - \eta) \quad (2.65)$$

$$N_3(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 + \eta) \quad (2.66)$$

$$N_4(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 + \eta) \quad (2.67)$$

The displacements are interpolated as,

$$u = \sum_{i=1}^4 N_i^e u_i; \quad v = \sum_{i=1}^4 N_i^e v_i \quad (2.68)$$

where u and v are the displacements at any point in the element and u_i and v_i the nodal displacements. The derivatives of $\frac{\partial}{\partial \xi}$, $\frac{\partial}{\partial \eta}$ can be found as,

$$\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \quad (2.69)$$

In matricial form,

$$\frac{\partial}{\partial \xi} = \mathbf{J} \frac{\partial}{\partial x} \quad (2.70)$$

where $\mathbf{J}$ is the *Jacobian* operator that relates the natural and global coordinates. The derivatives with respect to the global coordinates can be obtained by,

$$\frac{\partial}{\partial x} = \mathbf{J}^{-1} \frac{\partial}{\partial \xi} \quad (2.71)$$

The stiffness matrix is obtained by

$$\mathbf{K} = \int_V \mathbf{B}^T(\xi, \eta) \mathbf{D} \mathbf{B}(\xi, \eta) dV \quad (2.72)$$

The strain displacement matrix $\mathbf{B}$ depends on the natural coordinates ξ, η . The volume of the element is given by,

$$dV = h \det \mathbf{J} d\xi d\eta \quad (2.73)$$

where $\det \mathbf{J}$ corresponds to the determinant of the Jacobian matrix and h to the thickness of the plate. The integral in the stiffness matrix is computed numerically by the Gauss quadrature in two dimensions. Considering,

$$\mathbf{K} = h \int_A \mathbf{F}(\xi, \eta) d\xi d\eta \quad (2.74)$$

where

$$\mathbf{F}(\xi, \eta) = \mathbf{B}^T(\xi, \eta) \mathbf{D} \mathbf{B}(\xi, \eta) \det \mathbf{J} \quad (2.75)$$

Performing the numerical computation,

$$\mathbf{K} = \sum_{i,j,k} \mathbf{F}_{i,j}(\xi, \eta) \omega_{i,j} \quad (2.76)$$

where $\mathbf{F}_{i,j}$ is a matrix dependent on the natural points ξ_i and η_j . In the four node element a 2x2 numerical integration is used for exact integration. The Gauss quadrature in two dimensions replaces the integration by a summation over Gauss points by

$$\int_{-1}^1 \int_{-1}^1 F(\xi, \eta) d\xi d\eta = \sum_{i=1}^p \sum_{j=1}^p \omega_i \omega_j F(\xi_i, \eta_j) \quad (2.77)$$

where p and q are the number of integration points in the ξ and η directions, respectively, and ω_i, ω_j the corresponding weights. The stiffness matrix can be computed as,

$$\mathbf{K} = \int_{-1}^1 \int_{-1}^1 h \mathbf{B}^T(\xi, \eta) \mathbf{D} \mathbf{B}(\xi, \eta) \det \mathbf{J} d\xi d\eta \quad (2.78)$$

Chapter 3

State of the Art

There is a growing interest in the application of Machine Learning (ML) in a wide spectre of domains, ranging from identifying spam in our emails to pivotal applications in health care for diagnosing medical conditions. The use of ML is the result of an exponential growth of computational power and available data, coupled with declining costs associated with training AI systems.

The field of computational mechanics involves finding approximate solutions by solving partial differential equations (PDEs). One of the numerical methods available is the Finite Element Method (FEM). The FEM can be employed to several fields, ranging from Solid Mechanics, Biomechanics, Material Science, Geomechanics, among others.

The solution process of Finite Element Analysis (FEA) is mathematically intensive as it involves solving simultaneous linear equations. Since the complexity of problems continues to escalate, so does the size of these equations, leading to an increase in computational resources and time. To tackle with this problem, data driven approaches have been applied to the field of computational mechanics.

The use of ML approaches combined with Finite Element is a particularly recent discovery and it emerged only in recent decades. The common interest is to improve the trade-off between computational accuracy and costs. Among ML algorithms, Artificial Neural Networks (ANNs) have demonstrated an improved performance, across a spectrum of applications within FEA, due to its universal approximation property.

Machine Learning techniques recently have emerged as effective finite element surrogates in addressing biomechanical problems. Researchers used this techniques to analyse the shear stresses in the Abdominal Aortic Aneurysm (Soudah et al., 2015), study the biomechanical behaviour of tissues ((Martínez-Martínez et al., 2017), (Lorente et al., 2017)), predict the mechanical displacement in long bones (Mouloudi et al., 2020), among others.

Soudah et al. (2015) developed two ANNs to analyse shear stresses in the Abdominal Aortic Aneurysms (AAA) in order to reduce the computational time used by FEM. The proposed ANN model exhibited an accuracy of 95% when compared to the FEM whereas the computational time is reduced from approximately 5 hours to 2 mins. The work of Liang et al. (2018) also demonstrated how DL can reduce the simulation time, particularly in computing the stress distribution

of the human thoracic aorta. According to the study, the error rates for average von Mises stress distribution and maximum von Mises stress were reported to be 0.492% and 0.891%, respectively.

In the work of [Gholipour and Arjmand \(2016\)](#) two ANNs were trained to predict the three-dimensional posture of the spine with data from an inertial tracking device in reaching and lifting activities. The results indicated that both ANNs were capable of predicting the outputs with a determination coefficient, $R^2 > 0.95$, and a root-mean-squared-error, $RMSE < 11^\circ$.

An approach to forecast the real time deformation in breast tissues was introduced by [Martínez-Martínez et al. \(2017\)](#), employing a trio of tree based models (decision tree, extremely randomized tree and random forest) that were trained with data from Finite Element simulations. The surrogate model aimed to predict real-time deformation of breast tissues under compression. [Lorente et al. \(2017\)](#) used a similar approach to predict the biomechanical behaviour of the human liver during the breathing process, which is crucial for guiding surgeons during interventions where it is critical to track this deformation.

A finite element surrogate model was developed by [Madani et al. \(2019\)](#) to predict the stress distribution in the aortic wall of patients with atherosclerosis. The model learns the complex, non-linear relationship between aortic input parameters - such as tissue geometry, composition, and arterial pressure - and the resulting stress distribution. Their results indicate that, given a specific geometry, the model can predict the peak von Mises stress magnitude in real time, with an average error of less than 10%, allowing a quick clinical analysis of the patient health. The work of [Soleimani et al. \(2023\)](#) also addresses the prediction of atherosclerosis through the development of ANNs using data from multi physics Finite Element Method simulations. The speed and accuracy of the prediction highlight the computational gain since the ANN model could accurately predict the appearance of atherosclerosis in a fraction of seconds while the FEM simulations would take at least 2 hours (this for 2D analysis).

[Mouloodi et al. \(2020\)](#) employed an ANN to predict the mechanical displacement in long bones from ex-vivo measurements of load, load exposure time and strains. For a better prediction of the peak values of cyclic displacement, a Gaussian noise component was introduced to the data points. The ANN successfully predicted the displacement, as shown with a determination coefficient $R^2 > 0.98$, and achieving this outcome 160 times faster compared to the Finite Element Method.

[Pais et al. \(2023b\)](#) applied Feed Forward Neural Networks to bone remodelling by gathering a dataset of density distribution comprising several geometries and load cases. After training the network, the trabecular distribution obtained was similar to the FEM analysis, as visualized in [Figure 3.1](#). Notably, the analysis time was significantly reduced, taking only 0.064s instead of 1020s required by the FEM.

In the study by [Moura et al. \(2024\)](#), a Machine Learning framework was developed to predict stresses on the pelvic floor muscles (PFM) during childbirth using data from Finite Element Method simulations. Four Machine Learning algorithms - Random Forest (RF), Extreme Gradient Boosting (XGBT), Support Vector Regression (SVR), and Artificial Neural Networks (ANNs)

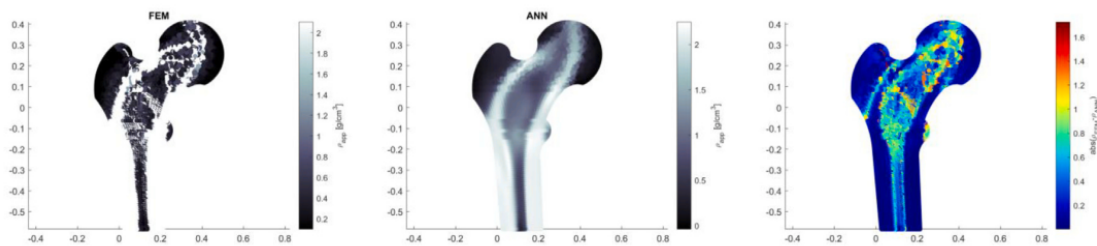


Figure 3.1: Neural Network performance compared with the FEM, for a particular bone geometry - Pais et al. (2023b)

- were employed to predict stresses at the moment of maximum muscle stretch and to predict multiple stresses throughout fetal descent.

The results indicated that the ANN outperformed the other models in the first case. In the second case, XGBT provided lower errors for 20 mm and 35 mm of fetal descent, while the ANN achieved better results for larger dimensions of fetal descent (50 mm and 65 mm).

Artificial Neural Networks (ANNs) can also be used to predict the die roll in the fine blanking process. Stanke et al. (2017) used data from two-dimensional axisymmetric FE simulations of a fine blanking process to train an ANN. The developed parametrized numerical model allowed for the automatic variation of process parameters such as punch edge, die edge radius or geometry, blank holder and counter punch force.

Atta et al. (2019) considered the geometric parameters (plate width, bolt diameter, distance between bolt axis and plate edge) to determine the five failure stages of double lap bolt joints using data from the Finite Element Analysis (FEA) to train the Artificial Neural Networks (ANNs). From the comparison of both, similar results were presented with the value of the correlation coefficient (R) being bigger than 0.99 for all the sets.

The stress field prediction in the field of computational Solid Mechanics using Deep Learning (DL) approaches is a current topic of research. Nie et al. (2020) used Convolutional Neural Networks (CNNs) to predict the stress fields in a 2D linear elastic cantilever beam structure. Two CNN architectures were used, a *single input channel network* (SCSNet) and a *multichannel input network* (StressNet). Jiang et al. (2021) proposed a conditional generative adversarial network (cGAN) called *StressGAN* to predict the von Mises stress of various solid structures for different geometries, loads and boundary conditions. Both of the authors reported that the *StressGAN* outperformed the *StressNet* model proposed in Nie et al. (2020) with a significant margin. Additionally, it was proved to be more accurate in critical areas.

More related to what is going to be done in this work, Ribeiro et al. (2021a) developed an Artificial Neural Network (ANN) to estimate the stress distribution in a plate with a central hole subjected to an uniaxial stress. The radius of the central hole and its central position (x,y), were the input variables of the ANN.

A database was created with a Finite Element Method (FEM) software for a given range of the radius of the plate and, the stress values were taken. For the case, 80% of the data was used for

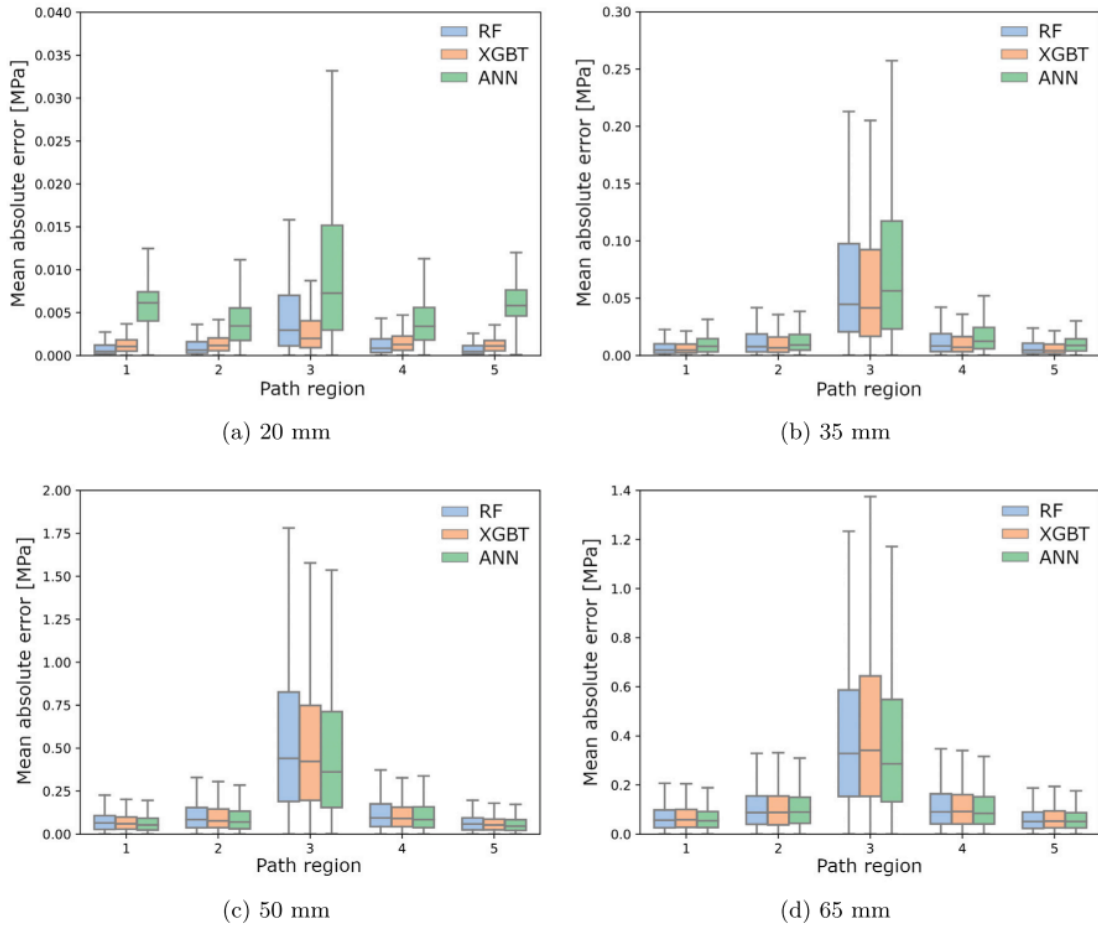


Figure 3.2: Mean Absolute Error (MAE) at four different levels of fetal descent - Moura et al. (2024)

training purposes while the other 20% was used for testing the ANN. The forecast was made for a hole radius between 25 and 50 mm. The author concluded that the stress map was very similar in terms of stress values and their distribution over the plate, as depicted in the Figure 3.3. For the two cases of radius equal to 25 mm and 50 mm, as illustrated in the Figure 3.3, the Mean Relative Error (MRE) is of 2.54% and 6.59%, respectively. In terms of time, and compared to the FEM software, the ANN is approximately 80 times faster in determining the stress field. With this, Ribeiro et al. (2021a) stated that the calculation of stress in the ANN instead of FEM software led to a significant reduction in time, thereby reducing the computational costs.

Similar findings were observed in the work of Ribeiro et al. (2021b), where an ANN was also used to predict the stresses but also the displacements of two different cases, a fixed plate subjected to a uniformly distributed pressure and a reinforced panel, fixed at its ends, subjected to internal pressure. As in the work of Ribeiro et al. (2021a), the most significant difference between the displacement and stress fields lies in the regions' contours. The author underscored the efficacy and swiftness of ANNs, highlighting their potential for a wide array of applications, especially in mechanical design, machine elements, and structural engineering.

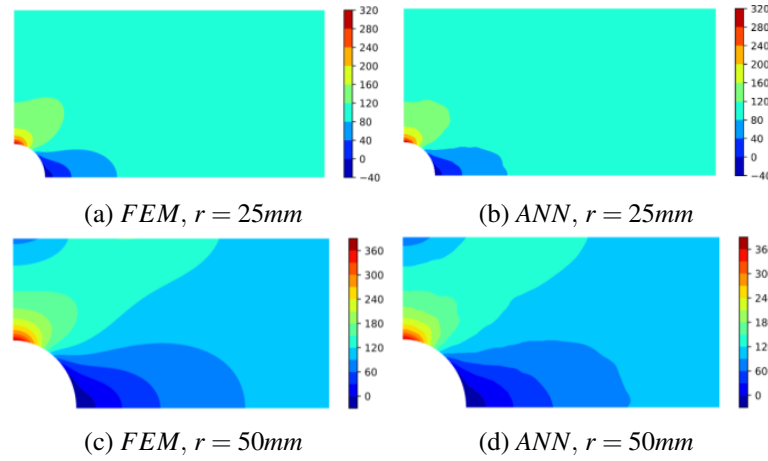


Figure 3.3: Stress Fields - Ribeiro et al. (2021a)

A guided wave-based Artificial Neural Network (ANN) model, proposed by Luca et al. (2021), was developed using the Finite Element Method to identify the location of damages. The ANN was initially trained on an aluminum plate and subsequently tested on a composite plate with various damage scenarios. The findings demonstrated that the ANN was highly accurate in detecting and localizing damages.

Sun et al. (2021) developed a framework for predicting the buckling and ultimate loads of composite hat-stiffened panels under in-plane shear through Artificial Neural Networks (ANNs). Firstly, a Finite Element Analysis was made to obtain the required data to train the model. Then, an autoencoder was employed to compress the original characteristics and establish back propagation and the generality of the ML model was verified. The results showed that the ANN effectively predicted the failure and buckling loads of the panel.

Ozkan and Erdemir (2021) investigated the Stress Concentration Factor (K_t) for circular and elliptical holes with bead reinforcement in an infinite panel under axial and biaxial loading states. The accuracy of K_t was validated through various methods and later an ANN was developed to determine K_t values and stress distributions based on dimensional ratios and loading conditions. The ANN models reproduced the values of K_t with high accuracy and in a fraction of time.

Badarinath et al. (2021) compared the performance of ML models, such as Decision Trees (XGBoost), Random Forest (RF) and Artificial Neural Networks (ANNs), in the estimation of stress distribution in a one dimensional beam structure. The author demonstrated that for the different models, the ANNs were superior in accurately predicting the values at any time and spatial location of the beam, as shown in Figure 3.4, according to the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R-squared values. This approach substantially reduces the computational cost, however it requires extensive datasets for optimal performance.

Similar to what was made, Hashemi et al. (2023) also studied the performance of various ML models as a surrogate FEM model. They predicted the time varying displacements on a two dimensional 31-bar planar truss structure. The authors concluded that the XGBoost regression was superior to the other three models (Random Forest, Adaboost regression DTs, DNNs) in obtaining

Model Performance Metric	ANNs	RF	XGBoost
Mean Absolute Error	2.87	6.55	15.47
Root Mean Squared Error	10.58	19.62	35.42
R-squared	0.9994	0.9981	0.9940

Figure 3.4: Performance of the ML models - [Badarinath et al. \(2021\)](#)

accurate results.

In the field of materials, more precisely composite materials, [Bhaduri et al. \(2022\)](#) used Convolutional Neural Networks (CNNs) to predict the von Mises stresses in a fiber reinforced composite. The U-Net architecture was able to accurately predict the stresses for the composites comprising 6, 10, 20 and 100 fibers. Similarly, [Shokrollahi et al. \(2023\)](#) used CNNs to predict the stress field maps for uniform, non uniform, irregularly shaped and a combination of these three fibrous structures. The same architecture was used and trained with data from FEM simulations. The CNN, based on the given architecture, demonstrated an exceptional behaviour in predicting the stress fields with structural similarity index scores (SSIM) of 0.997. Both of the conducted studies affirmed that the DL models achieved an excellent accuracy in determining the stress fields.

[Bolandi et al. \(2022\)](#) used CNNs to predict the stress distribution across a loaded 2D linear elastic steel plate for a given geometry, loads, and boundary conditions as inputs. The results were compared to the ones obtained with the Finite Element simulations, demonstrating accuracy and reliability with a Mean Absolute Error (MAE) of 0.9% and an absolute maximum error of 0.46%.

In the area of stress recovery, [Khoei et al. \(2020\)](#) introduced a method to estimate the recovered stress field in areas with high stress gradients. By using an Artificial Neural Network (ANN) as the approximation function, their approach yielded more accurate predictions of the stress distribution, particularly in regions with high stress gradients. In their work, three separate ANN models were trained to predict each stress component individually. Similarly, and within the same domain, [Saikia et al. \(2023\)](#) applied ANNs to predict stresses comprehensively across the entire domain, considering different mesh sizes. Their study focused on Cook's skew beam problem and employed the Superconvergent Patch Recovery (SPR) technique, which approximates stress values at Gauss points using polynomials. Unlike [Khoei et al. \(2020\)](#), who concentrated only on high-stress regions, [Saikia et al. \(2023\)](#) took a broader approach, examining stress distribution across the full domain.

Investigations have been conducted in the realm of the structural response of masonry arches. [Motsa et al. \(2023\)](#) used ANNs to forecast the behaviour of masonry arches, training the ANNs with data derived from Finite Element Analysis (FEA). [Thango et al. \(2024\)](#) treaded a similar path as [Motsa et al. \(2023\)](#), employing ANNs to predict the deflection of masonry walls under blast loading. Both of the authors, concluded that the ANNs are a reliable tool to substitute for the dynamic finite element simulations.

In the domain of geotechnical engineering analysis, [Gao et al. \(2020\)](#) used a DL architecture, called the enhance-and-split feature capsule network embedded in fully convolutional neural net-

works (ES-Caps-FCN) to predict in situ stresses (intrinsic stress of the crust and rock formations in their original state). To train the algorithm, the authors used fast Lagrangian analysis based on the finite difference method. The proposed algorithm achieved fast and accurate results, being a feasible option to be used.

Hsiao et al. (2022) proposed a pretrained model to estimate the safety factor and the trace of the slope slip surface by using ML techniques, such as ANNs and CNNs. Both models were fed with data from finite element simulations. CNNs presented a better performance with increased training samples, which show their ability to capture spatial relations effectively.

The literature show that there was an exponential interest in the application of Machine Learning (ML) techniques, particularly Artificial Neural Networks (ANNs), within the field of computational mechanics, specially in relation to the Finite Element Method (FEM), over the past decade. This trend is largely driven by remarkable improvements and advancements in the Artificial Intelligence domain in recent years.

The reviewed studies highlight how ML techniques, specially ANNs, have been successfully used as surrogates to traditional FEM across various domains. The applications show the ability of the ML models to achieve high accuracy while drastically reducing the computational time and resources. The examples range from predicting stress and strain distributions in structural components to estimating biomechanical behaviour in medical contexts, demonstrating the versatility of ML approaches.

Overall, the approach followed through this study adhered to a well defined methodological framework, applicable to almost all cases analysed. In the beginning, the problem was analysed, along with the necessary data, such as the displacements, stresses and strains which are crucial for predicting the mechanical response of a system. Next, a dataset was generated for training, typically through simulations using the Finite Element Method (FEM). Afterward, the neural network model was selected, followed by hyperparameter tuning to optimize the performance of the model. Once the optimal configuration of hyperparameters was determined, the model was trained using the available data, and its performance was validated in terms of accuracy in predicting the desired results. This methodological framework provided a structured and consistent analysis of the various approaches, ensuring that each model was properly tuned and rigorously evaluated.

Chapter 4

Plate with a central hole

This chapter presents a well known benchmark problem in Solid Mechanics: a plate with a central hole subjected to a distributed load. A similar structured approach to that outlined in the previous chapter was applied. First, the problem of the plate was analysed and, subsequently, FEM simulations were conducted to generate the necessary dataset for training the neural network. The neural network model was then selected, and the influence of hyperparameters was explored, in line with the hyperparameter tuning methodology applied throughout the study.

For predicting the displacements, the network was trained with data from the nodes. In the second part of the analysis, the focus was on predicting stresses and strains at the integration points, with the values calculated at these points gathered for training and validation. Finally, the networks ability to predict the stress and strain values at the nodes was tested, assessing the model's capability to generalize beyond the integration points

4.1 Problem Description

The case study of this work presents a classical example in Solid Mechanics, which is a two dimensional plate with a central hole subjected to a distributed load on both sides of the plate, as shown in Figure 4.1. The plate variables are its length L , width W and the radius r of the hole.

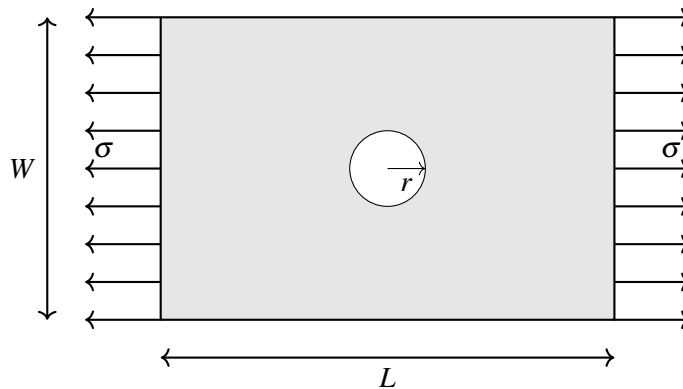


Figure 4.1: Plate with central hole

The plate is subjected to a uniformly distributed load on both sides, represented by the stress σ . To ensure consistency in the applied stress across all plates, a unit force is divided by the width (W) of the plate. Since the analysis is conducted within the elastic regime, the calculated displacements can be scaled for different applied forces. This normalization of the data not only ensures uniformity but also facilitates the training of the neural network by simplifying the learning process.

The force to which the plate is subjected, will cause a deformation on it. Several parameters within the plate, such as the displacements, stress and strain, will be analysed through the use of the Finite Element Method (FEM). Subsequently, and with the data generated from the FEM, an Artificial Neural Network (ANN) will be selected and trained to predict the displacements, stresses and strains among the plate, both with and without the presence of outliers in the FEM data.

Once the neural network is trained, its accuracy in predicting the already mentioned parameters will be evaluated by comparing the results to the ones obtained from the FEM analysis. This comparison will also assess the influence of outliers on the neural network's performance.

Within the Artificial Neural Network, several internal parameters will be studied and adjusted in order to obtain the best possible results, with the least error in predicting the desired output. The weight that each of this internal parameters have in the predictions will be studied in depth.

4.2 Methodology

4.2.1 FEM Analysis

The plate, which is under a force, is going to be analysed using the Finite Element Method. The desirable output variables are the displacement, the stress and the strain. Since the plate is symmetric in the x and y axis, the problem can be simplified by only analysing one quarter of the plate. The FEM software used for this analysis was *Abaqus*, version 2022.

To generate a dataset for training the ANN, it is necessary to perform multiple simulations. However, executing various simulations one by one would be highly time consuming. Thus, automating this process became imperative. One way to automate this process is through the use of *Python* scripts in *Abaqus*. *Python* scripting enables to run a series of simulations with varying parameters and the collection of the resulting data.

Prior to delving into *Python* scripting, a CSV file that contains the values of the varying parameters of the plate had to be created. This file encompasses all the values of length L , width W and radius r to be used in the simulations performed in *Python* scripting. The length and width range from 1 to 5 mm, with increments of 0.5 mm, whereas the radius varies from 0.05 mm to the minimum value of the length or width minus 0.1 mm. The radius increment value between each simulation is 0.05 mm. With this, 3533 unique combinations of parameters were obtained, resulting in an equivalent number of simulations to be performed in *Abaqus*. The code pertaining to this CSV file is provided in Annex A.1.

Using *Python* scripting in *Abaqus* software, two-dimensional (2D) plate models were generated. For each plate model created, the script assigned the plate's length, width and radius with the corresponding row value in the CSV file. With these values, the sketch was made and the part was created, considering a two dimensional deformable body. This script is also provided in Annex A.2.

The code generated the geometries and assigned the linear elastic material properties, with a Young's Modulus (E) equal to 1 MPa and a Poisson's ratio (ν) equal to 0.3. A meshing size of 0.05 mm with two dimensional plane stress elements (CPS4) was chosen. The CPS4, 4-node quadrilateral element, is used for modelling problems under plane stress conditions, such as the case under study.

The application of symmetric boundary conditions allowed to reduce the size of the problem, taking less amount of memory and time to achieve the results. Regarding loads, a uniform distributed load is applied. A concentrated force, given by 1 N divided by the number of nodes in the width of the plate, is applied to each node.

Regarding the mesh discretization density, a mesh size of 0.05 was defined. The meshed plate geometry as well as its boundary conditions can be seen in the following Figure 4.2.

Once the pre processing phase is complete it is time to pass to the post processing phase, where the desired results are obtained. For each model simulation the script generates three CSV files, each one with the desired output. One .csv file exhibit the displacement of the nodes in the x and

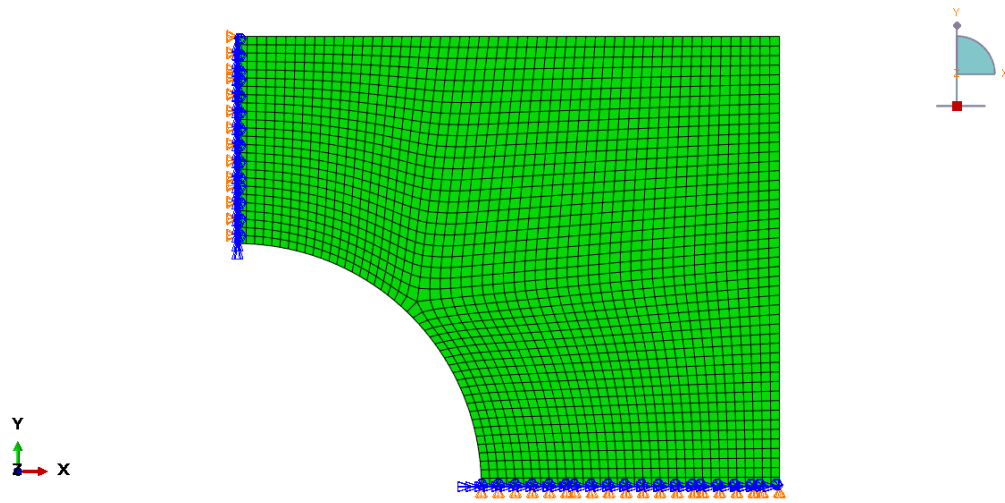


Figure 4.2: Plate Boundary Conditions

y direction, other the stress and strain at each node and another shows the stress and strain at the integration points.

4.2.2 ANN Training

With all the data from the simulations extracted, the next step was to proceed with the development of an Artificial Neural Network (ANN). The training of the neural network was done with the use of *Python*.

The data gathered from the FEM simulations allowed to create a dataset for the training of the neural network. For each simulation, three *.csv* files were generated, each containing different data outputs: node displacements, stress and strain values for each node, and stress and strain values for the integration points. This data will be used for training and testing the neural network to make predictions.

The first outputs to be studied in the artificial neural network were the node displacements, both in the x and y directions. Subsequently, the network was trained to predict stresses and strains at the integration points, followed by stresses and strains at the nodes.

The data used to train the network for each variable is contained in the corresponding *.csv* files. The comma-separated values (csv) file format for the node displacements is illustrated in Figure 4.3.

It presents the following columns: *ID of the node*, its respective x and y coordinates, the *ID of the node* once more, and its corresponding displacement u (in x direction) and v (in y direction). Following this, there are the columns corresponding to the features of the plate: its *length*, *width* and *radius*. The other *.csv* files present a similar structure, with the key difference that for each one there are different parameters to be studied.

The decision to train each parameter separately from another was taken because training separate models for each output is more effective, as it allows each model to specialize in predicting

	A	B	C	D	E	F	G	H	I
1	1	0.0	0.95		1	4.8692463 -388.6174	1.0	1.0	0.9
2	2	0.0	0.9		2	2.4411737 -387.7898	1.0	1.0	0.9
3	3	0.6363961	0.6363961		3	152.49892 -149.0697	1.0	1.0	0.9
4	4	0.9136396	0.9136396		4	17.390268 -14.00422	1.0	1.0	0.9
5	5	0.9	0.0		5	399.27994 -2.316789	1.0	1.0	0.9
6	6	0.95	0.0		6	400.02982 0.0	1.0	1.0	0.9
7	7	1.0	0.0		7	399.02914 2.3139446	1.0	1.0	0.9
8	8	1.0	0.9411765		8	4.009946 28.280897	1.0	1.0	0.9
9	9	1.0	1.0		9	-24.71899 28.241144	1.0	1.0	0.9
10	10	0.9411765	1.0		10	-24.82039 -0.542983	1.0	1.0	0.9
11	11	0.0	1.0		11	-1.928097 -387.7398	1.0	1.0	0.9
12	12	0.0441609	0.8989159		12	5.6772637 -385.4373	1.0	1.0	0.9
13	13	0.0882154	0.8956662		13	11.389376 -378.9817	1.0	1.0	0.9
14	14	0.1320574	0.8902588		14	17.247961 -368.8478	1.0	1.0	0.9
15	15	0.1755812	0.8827067		15	23.388899 -355.7810	1.0	1.0	0.9
16	16	0.2186821	0.8730281		16	29.976978 -340.5547	1.0	1.0	0.9
17	17	0.2612562	0.8612463		17	37.158047 -323.8451	1.0	1.0	0.9
18	18	0.3032008	0.8473896		18	45.04095 -306.1949	1.0	1.0	0.9
19	19	0.3444151	0.8314916		19	53.697136 -288.0186	1.0	1.0	0.9
20	20	0.3847995	0.8135903		20	63.16801 -269.6279	1.0	1.0	0.9
21	21	0.4242570	0.7937291		21	73.47309 -251.2564	1.0	1.0	0.9
22	22	0.4626924	0.7719557		22	84.6169 -233.0816	1.0	1.0	0.9
23	23	0.5000132	0.7483226		23	96.59357 -215.2400	1.0	1.0	0.9
24	24	0.5361293	0.7228868		24	109.39011 -197.8392	1.0	1.0	0.9
25	25	0.5709539	0.6957094		25	122.98830 -180.9655	1.0	1.0	0.9
26	26	0.6044031	0.666856		26	137.36588 -164.6897	1.0	1.0	0.9
27	27	0.8566442	0.9190109		27	14.699925 -41.89662	1.0	1.0	0.9

Figure 4.3: .csv file containing the node displacements

its respective output without interference. This specialization leads to better performance and accuracy, as each model can fully focus on learning the specific patterns and features relevant to its designated output.

The steps to perform the training of the neural network are enunciated. The network imports the data from all the corresponding CSV's to a matrix and splits it into training and testing sets using the *random.choice()* function. It was defined that 70% of the data will be used for training purposes, while the remaining 30% was allocated for testing.

Once the data had been split for training and testing, both the data is normalized. Normalization is a crucial pre processing step in Machine Learning, and it is done with the intention of improving the performance of the model. By doing this, it ensures that all the inputs features and targets (or outputs) are on a similar scale, making it easier for the network to learn the relation between the input features and the target, which can be the displacement, stress or strain. The *MinMaxScaler()* function, from *scikit-learn*, was used to normalize the data. This function scales each feature so that the values have a given minimum and maximum value, of 0 and 1, respectively. The expression to scale the feature values is given by :

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (4.1)$$

When the input features and output are normalized, the parameters of the network need to be defined before the training process. This parameters, also known as *hyperparameters*, are the internal parameters of the model and, cannot be learned from the training phase and, as a

result, are fixed before the training. The selection of appropriate *hyperparameters* is a crucial step as it directly affects the performance and behaviour of a machine learning model. The process of finding the best combination of hyperparameters for the neural network to achieve its best performance is called *Hyperparameter Tuning*.

The objective of the *Hyperparameter Tuning* is to reach the best model. This analysis contributes to the influence of each aspect individually but, as they all interact (e.g. a better batch size with a worse activation function), this leads, for example, to the analysis tending to conclude that the activation function is better.

The tune process consists of exploring different combinations of hyperparameters in order to identify the best set of parameters that maximize the model performance on a validation set. In the created model, the hyperparameters to be tuned are the activation function, the number of hidden layers, the number of neurons in each hidden layer, the learning rate, and the batch size.

The hyperparameter values considered are as follows:

- **Number of hidden layers:** 10, 20, 30
- **Number of neurons in each hidden layer:** 8, 16, 32
- **Learning rate:** 0.01, 0.0001
- **Batch size:** 32768, 65536

The number of epochs, which corresponds to when all the training data is processed once, was defined at 1000. A callback from *Keras*, called *EarlyStopping*, was used in order to improve the performance of the model. The *EarlyStopping* callback allows to specify the performance measure to monitor during training and, when the monitored metric stops improving for a given number of epochs, it stops the training. For this case, the monitored metric was the "*loss*" function and as it is intended to decrease and achieve a minimum, the "*min*" mode was chosen where the training will stop when the "*loss*" function stops decreasing. The *patience*, which is the number of epochs with no improvement after which the training will be stopped, was defined at 100. With the implementation of the *EarlyStopping* callback, it was possible to conserve the computational resources by terminating the training process when no improvement was seen.

After this procedure, the network was put to the training process and the results were validated on the validation set. For each trained model with the different parameters, a file was created with the history of the training and validation error along the epochs. At the end, the model with the best performance i.e. with the minimum loss is saved and used to make predictions with the testing features.

Before training the neural network, it was observed that some of the output values were unusually high for certain values of length, width, and radius of the plate. Considering these as potential outliers, which could complicate the training process, the model will be trained both with and without these outliers in order to evaluate their influence on the results.

The outliers are identified by unusually high values in outputs like displacement, stress or strain, that occur when the hole in the plate is disproportionately large relative to the plate's dimensions. Specifically, if the hole radius exceeds 75% of either the plate's length or width, it is considered to be an *outlier*.

The trained neural network utilized various hyperparameters, activation functions, and loss functions. The influence of these parameters was thoroughly studied and reflected in the results.

The code relative to the neural network without outliers is presented in Annex A.3. For the neural network with the presence of outliers, the code is presented in Annex A.4.

4.3 Results

This section presents the results achieved by the neural network. The network was trained to predict various outputs, including node displacements, stress and strain at integration points and, stress and strain at the nodes. The influence of hyperparameters on the neural network's performance is considered, as well as the impact of *outliers* on the model's performance.

It is important to clarify that all references to "error" throughout this section pertain to the absolute error, which is defined as the difference between the neural network's prediction and the corresponding solution from the Finite Element Method (FEM).

4.3.1 Node Displacements

The first outputs to be studied in the neural network were the node displacements of the plate. These displacements include the displacements in both the x direction and y direction.

4.3.1.1 Node Displacement - x direction

Initially, the network was trained without outliers, using the Mean Absolute Error (MAE) as the loss function. The results of the displacement prediction u (in x direction) are shown in the following Figure 4.4.

The factor that shows a clear influence is the number of hidden layers, as evidenced by the non overlapping standard deviations shown in Figure 4.4b. The validation loss increases sharply as the number of hidden layers increases, indicating that 10 hidden layers outperform 20, and 20 layers perform better than 30.

The learning rate also demonstrates a significant influence on the model's performance. For a learning rate of 0.0001, the model performs considerably better, exhibiting a much smaller validation loss compared to a learning rate of 0.01. This difference is clearly illustrated in Figure 4.4e.

As previously mentioned, the analysis in Figure 4.4 did not consider the presence of outliers in the displacement u . To address this, a similar analysis was conducted with outliers included, and the results are presented in Figure 4.5.

From Figure 4.5, some key observations can be extracted:

- **Activation Function:** When the network encounters outliers, the *ReLU* activation function yields better validation loss results compared to the *tanh* activation function.
- **Number of Hidden Layers:** Deriving definitive conclusions about the optimal number of hidden layers is challenging, as the validation loss values vary, as shown in Figure 4.5b.
- **Hidden Layer Size:** A configuration of 8 neurons per hidden layer yields lower losses compared to using 16 or 32 neurons.
- **Batch Size:** Larger batch sizes correlate with smaller losses.

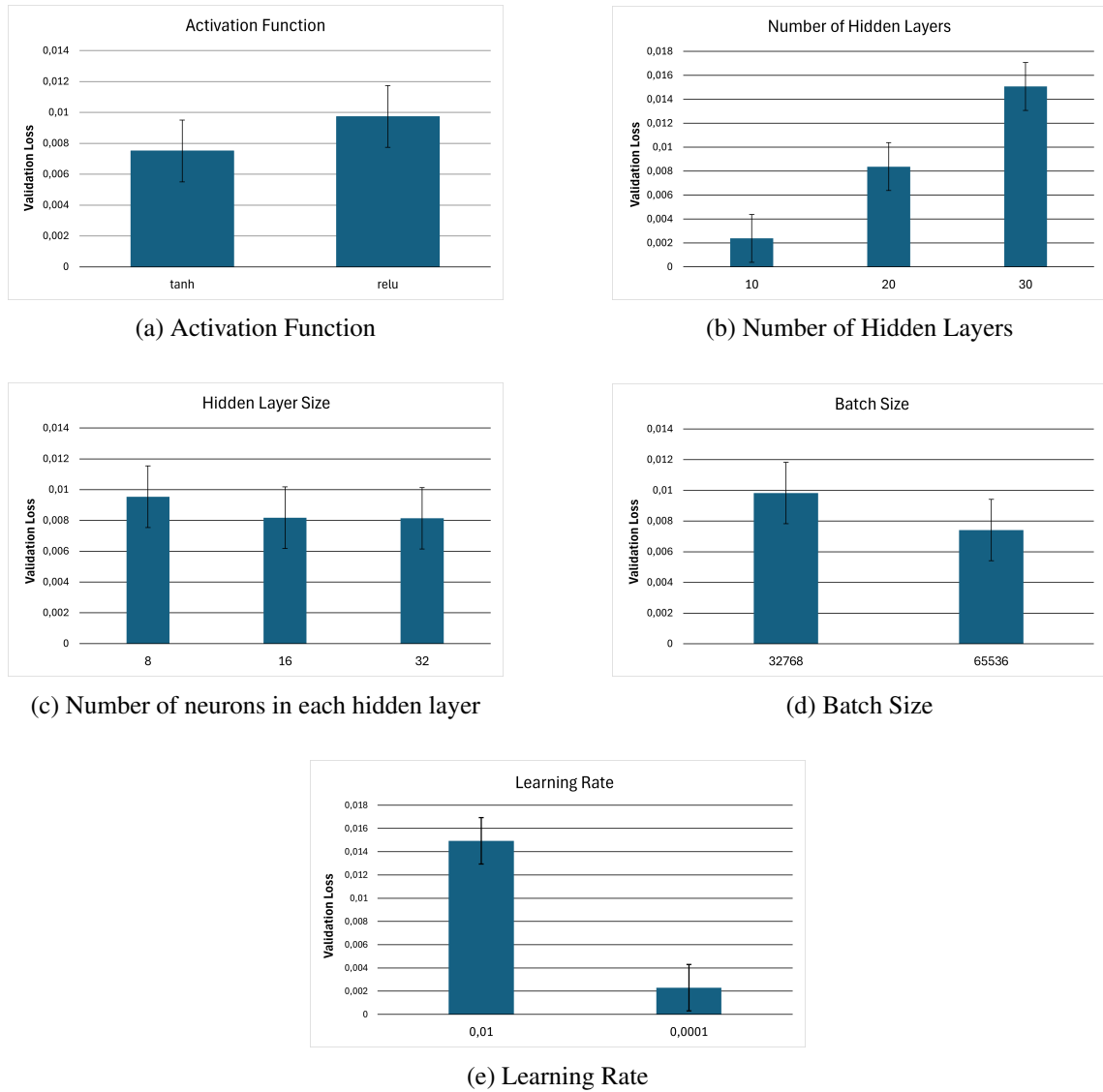


Figure 4.4: Sensitivity analysis of hyperparameters for the ANN model - Without Outliers

- **Learning Rate:** The most significant differences are observed in the learning rate; a learning rate of 0.0001 substantially reduces the loss, resulting in much better performance.

When comparing the performance of networks with and without outliers, it seems that the presence of outliers generally results in smaller losses. This occurs because the additional data includes more extreme values of displacement, which can artificially lower the loss metric by increasing the diversity of information the network learns from. As a result, the loss values cannot be directly compared between both datasets in order to have a fair comparison.

The performance of the best models, featuring the validation MAE along the epochs, was compared during training, both with and without the presence of outliers. For the Figure 4.6a, which depicts the model without outliers using the *tanh* activation function, the model quickly reduces the validation MAE to a very low value within the first 200 epochs and then continues to

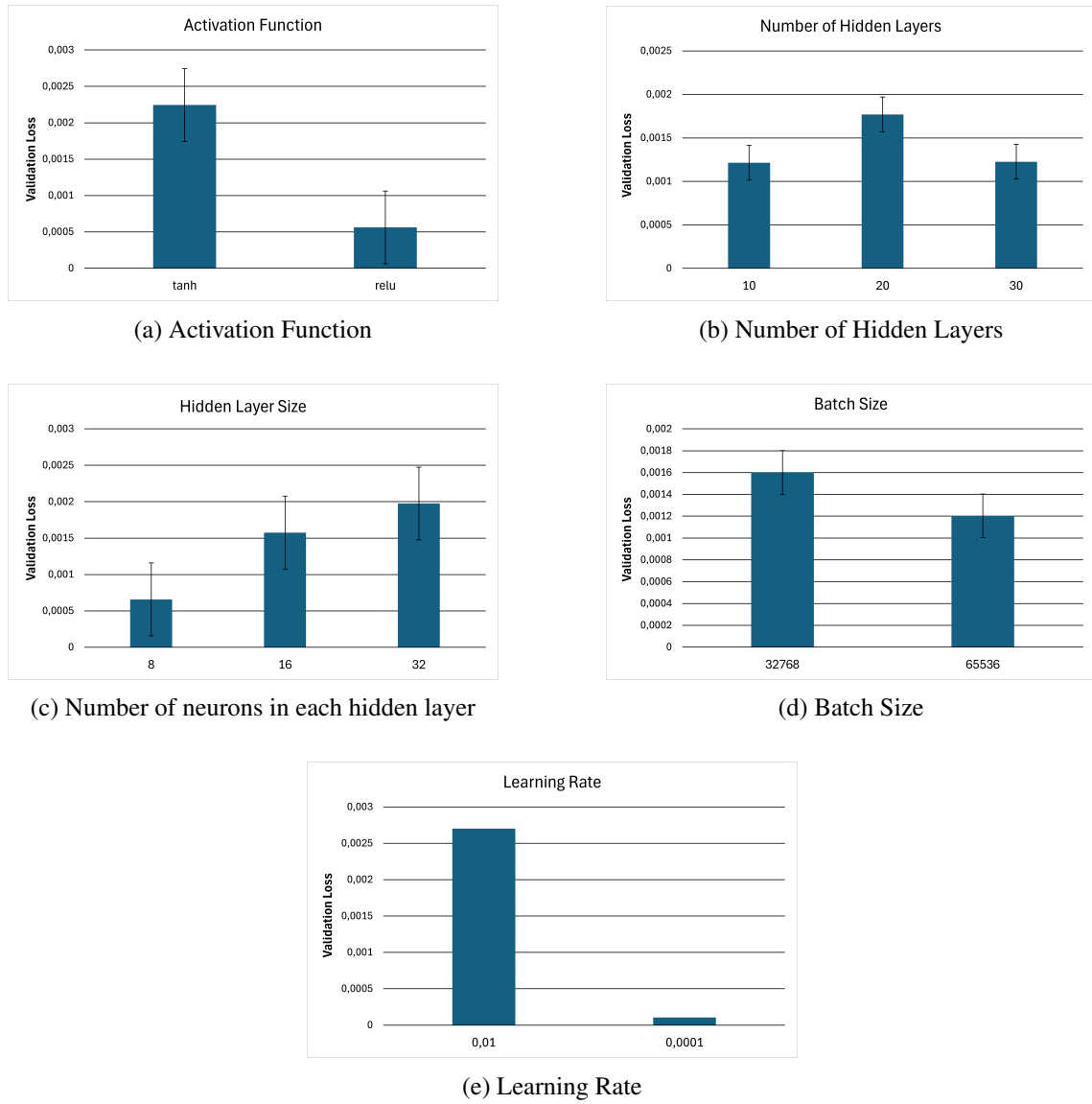
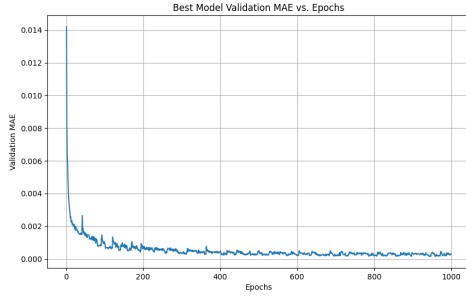


Figure 4.5: Sensitivity analysis of hyperparameters for the ANN model - With Outliers

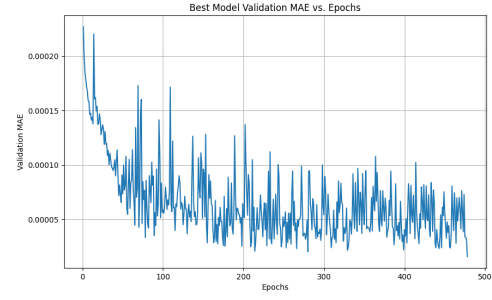
improve gradually, stabilising around a minimal value. Despite some minor fluctuations, the error generally remains stable after the initial drop. This strong overall performance indicates that the model learns effectively without the presence of outliers.

In contrast, for the model with outliers and using the *ReLU* activation function, depicted in Figure 4.6b, there are significant fluctuations in the validation MAE throughout the training process. This suggests instability and difficulty in learning due to the presence of outliers, indicating that the model's performance is adversely affected by them.

Although the model with outliers exhibits a smaller validation loss, this reduction is artificially induced by the presence of outliers, as previously said, leading to a considerably more unstable training process. The presence of outliers introduces high variance in the error distribution, causing the model's loss to fluctuate significantly during training. Consequently, when the model makes



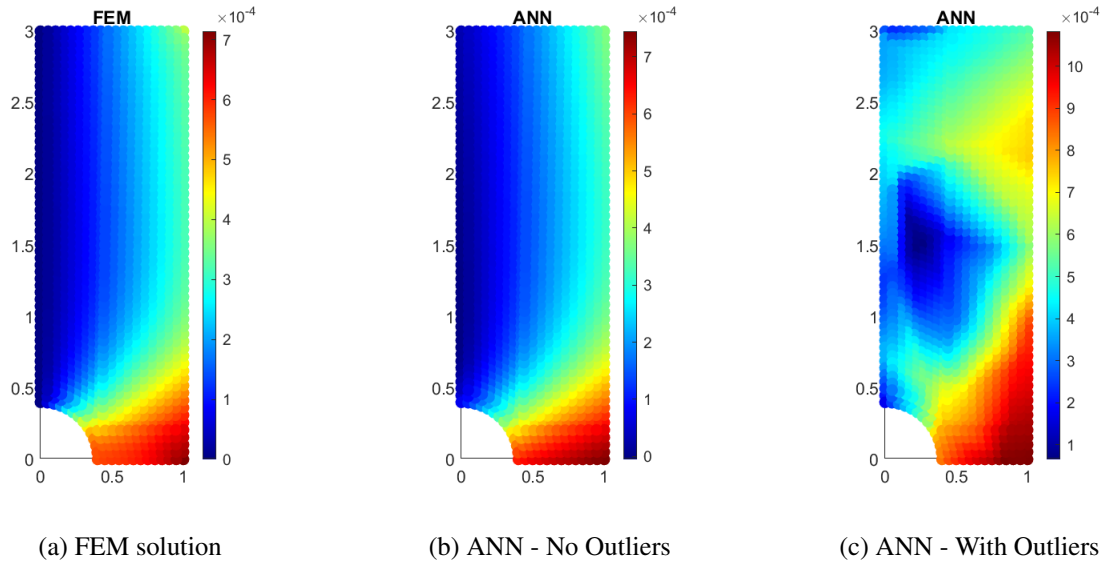
(a) Best Validation MAE across Epochs - Without Outliers



(b) Best Validation MAE across Epochs - With Outliers

Figure 4.6: Validation MAE across Epochs

predictions on the testing data, it struggles to accurately predict the displacement patterns, as illustrated in Figure 4.7. The code used to visualize these predictions is provided in Annex A.5.



(a) FEM solution

(b) ANN - No Outliers

(c) ANN - With Outliers

Figure 4.7: Different model predictions of the displacement in x direction

The model without outliers and using the *tanh* activation function predicts the displacements with remarkable accuracy, as shown in Figure 4.7b. In contrast, the model with outliers and employing the *ReLU* activation function faces difficulty in predicting the displacements in the x direction.

The optimal model for predicting the displacements in the x direction, achieved by excluding outliers and using the *tanh* activation, is characterised by the following hyperparameters that yield the lowest loss:

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 20

- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

4.3.1.2 Node Displacements - y direction

After analysing the displacement in the x direction, the focus shifted to studying the displacement in the y direction. The same methodology was applied, with the key difference being that the target data, representing the displacement v , corresponds now to a different column in the .csv file. The input features for the network remained the same.

Initially, the network was trained without outliers, utilising the Mean Absolute Error (MAE) as the loss function. The evaluation of hyperparameters in predicting the displacement v (in the y direction) is presented in Figure 4.8.

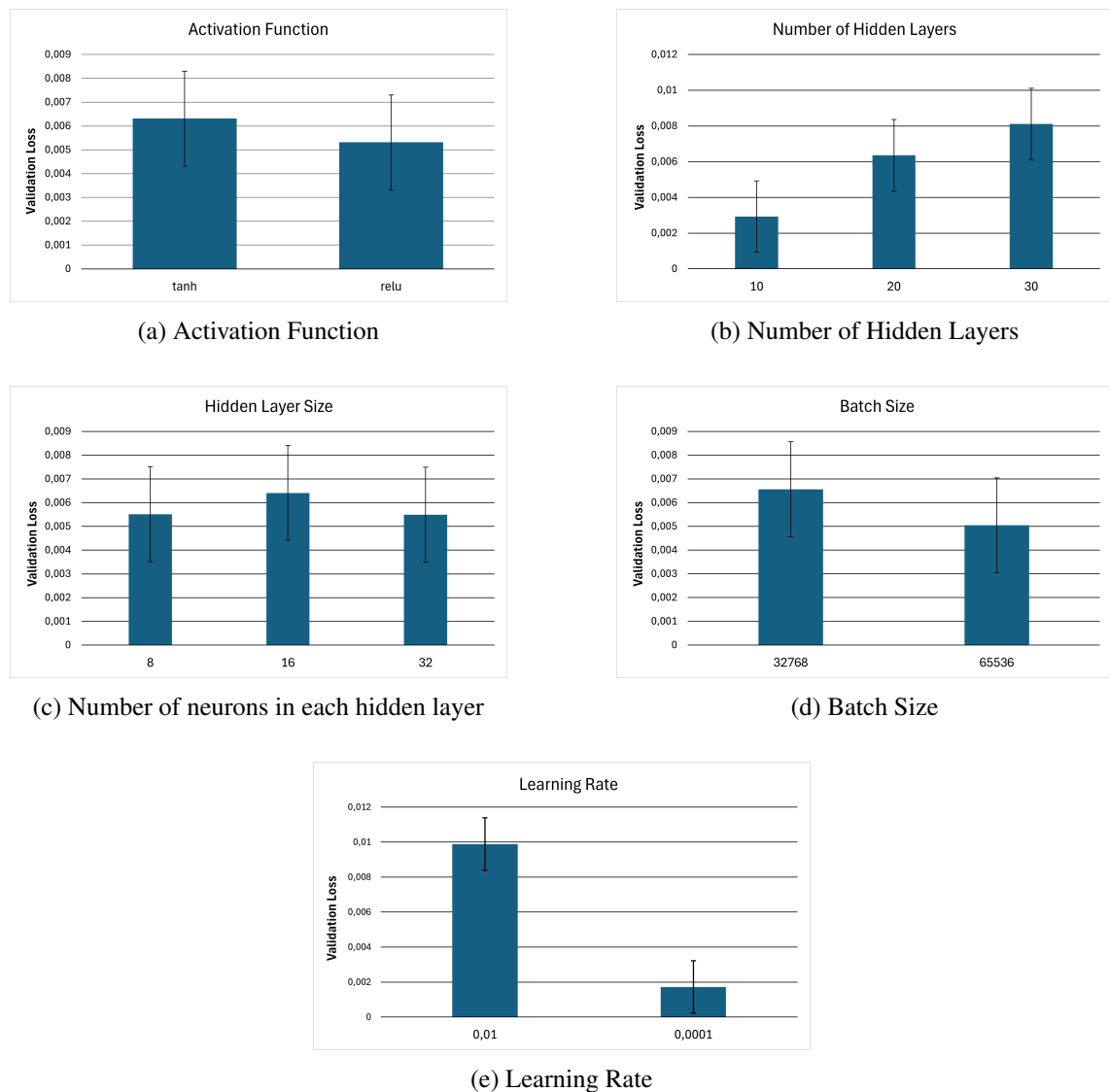


Figure 4.8: Hyperparameter sensitivity analysis for the ANN model - Without Outliers

The results show that the influence of the number of hidden layers and the learning rate on the displacement prediction v (in the y direction) closely mirrors the effects observed in the x direction.

Specifically, using fewer hidden layers produces more accurate predictions, and a learning rate of 0.0001 performs significantly better than a rate of 0.01.

In the subsequent Figure 4.9, outliers are included in order to analyse their impact and extract relevant insights on the influence of their presence.

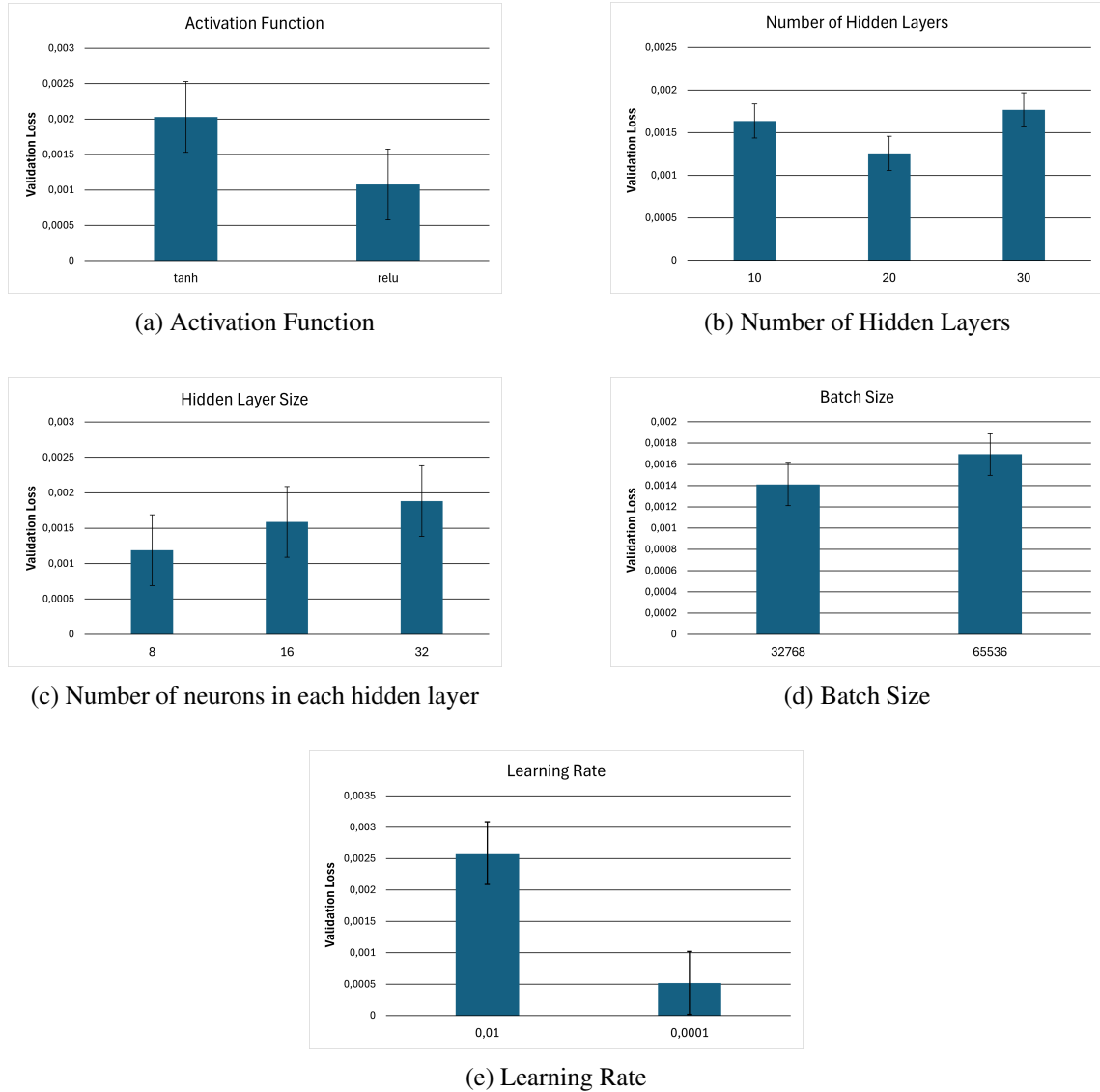


Figure 4.9: Hyperparameter sensitivity analysis for the ANN model - With Outliers

The analysis of Figure 4.9 reveals that the *ReLU* activation function performs better when outliers are present, and using 20 hidden layers yields better results compared to 10 or 30 layers. However, the most notable factor is the learning rate, with a value of 0.0001 consistently producing significantly lower validation loss on average.

From the comparison of the performance of the best models for the displacement in the y direction, represented in Figure 4.10, several observations can be made regarding the impact of activation functions and the presence of outliers on the model performance:

1. **Model Without Outliers using *tanh*** (Figure 4.10a):

- The *tanh* activation function handles the training process smoothly, resulting in strong overall performance.
- The validation MAE decreases rapidly within the first few epochs and then stabilizes with minor fluctuations, indicating consistent learning and effective convergence.

2. **Model With Outliers using *tanh*** (Figure 4.10c):

- When outliers are present, the *tanh* activation function shows a sharp initial decline in validation MAE, but the training process becomes unstable with considerable fluctuations throughout.
- These fluctuations suggest that outliers introduce variability, making it difficult for the model to converge smoothly and maintain consistent performance.

3. **Model Without Outliers using *ReLU*** (Figure 4.10b):

- The *ReLU* activation function also effectively learns without outliers, but it introduces significant fluctuations in the validation MAE.
- Compared to *tanh*, *ReLU* results in a more unstable training process, although it eventually reduces the MAE.

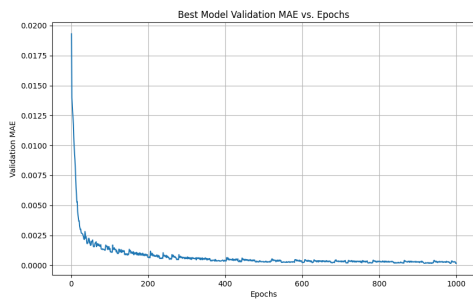
4. **Model With Outliers using *ReLU*** (Figure 4.10d):

- The validation MAE graph is peculiar, starting high and exhibiting a sharp, linear decrease to zero abruptly. The sharp drop suggests potential issues such as overfitting.

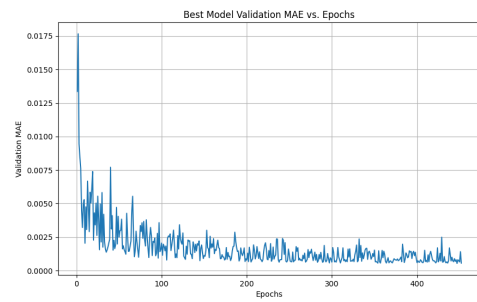
It is important to note that while the u displacements are predominantly positive, the v displacements are almost entirely negative. The results for the four best models, as introduced in Figure 4.10, are compared and analysed for the same length, width, and radius of the plate.

For the models without outliers, predictions were made using different activation functions, namely *tanh* and *ReLU*. These predictions were conducted on a plate with dimensions of 1.5 x 1.5. Figure 4.11 allows for a comparison and evaluation of the neural network's v displacement predictions for the different activation functions against the Finite Element Method (FEM) solution.

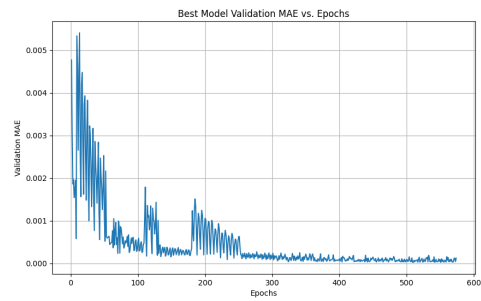
Analysing the Figure 4.11, it is evident that both activation functions generally perform well in predicting the displacement v . However, when comparing these predictions to the FEM solution and examining the error plots, it becomes clear that, as indicated by the Mean Absolute Error (MAE) graph across epochs, the *tanh* activation function provides more accurate predictions than the *ReLU* activation function, with a significantly lower error value.



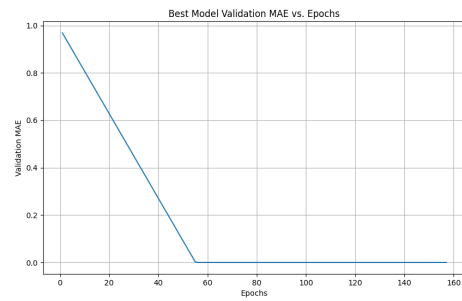
(a) Best Validation MAE across Epochs - No Outliers (tanh)



(b) Best Validation MAE across Epochs - No Outliers (ReLU)

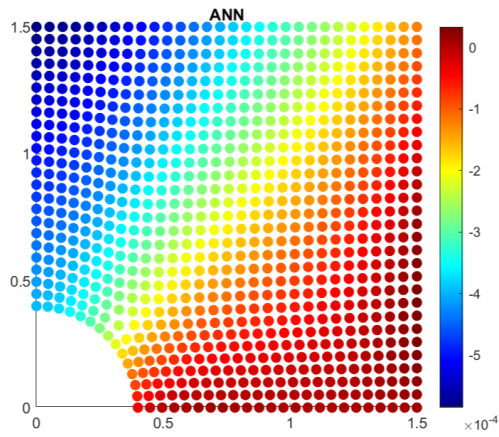
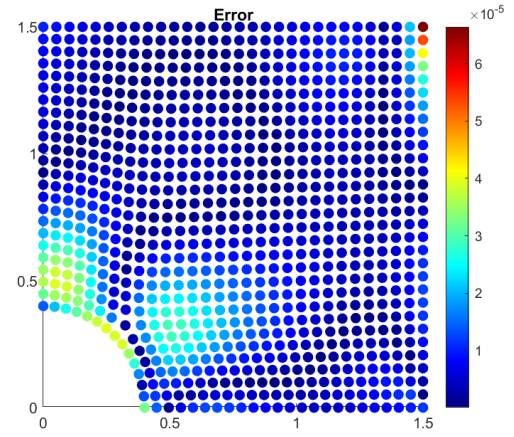
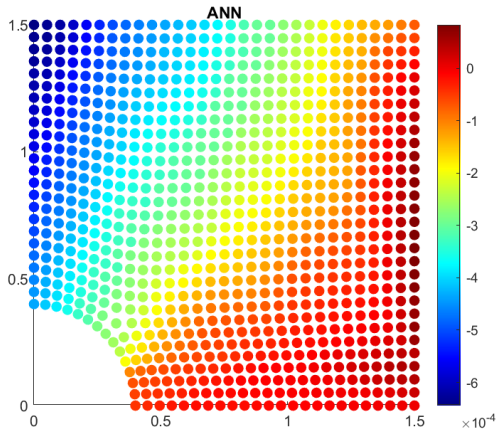
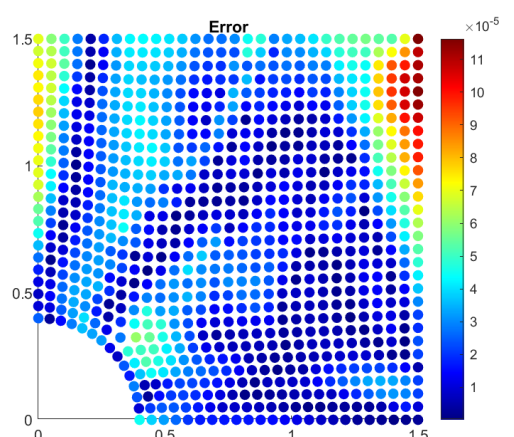
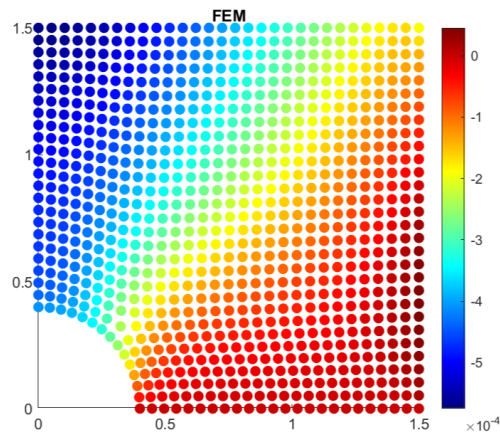


(c) Best Validation MAE across Epochs - With Outliers (tanh)



(d) Best Validation MAE across Epochs - With Outliers (ReLU)

Figure 4.10: Best Validation MAE across Epochs

(a) ANN - $\tanh$ (b) Error - $\tanh$ (c) ANN - ReLU (d) Error - ReLU 

(e) FEM Solution

Figure 4.11: Comparison of FEM and ANN solutions for displacement v - Without Outliers

For the models with outliers, predictions were again made using both *tanh* and *ReLU* activation functions. However, due to the *ReLU* function producing anomalous and unrealistic results, these are excluded from the presentation. The predictions, conducted on a plate with dimensions of 1.5 x 1.5, are visualized in Figure 4.12 for the *tanh* activation function.

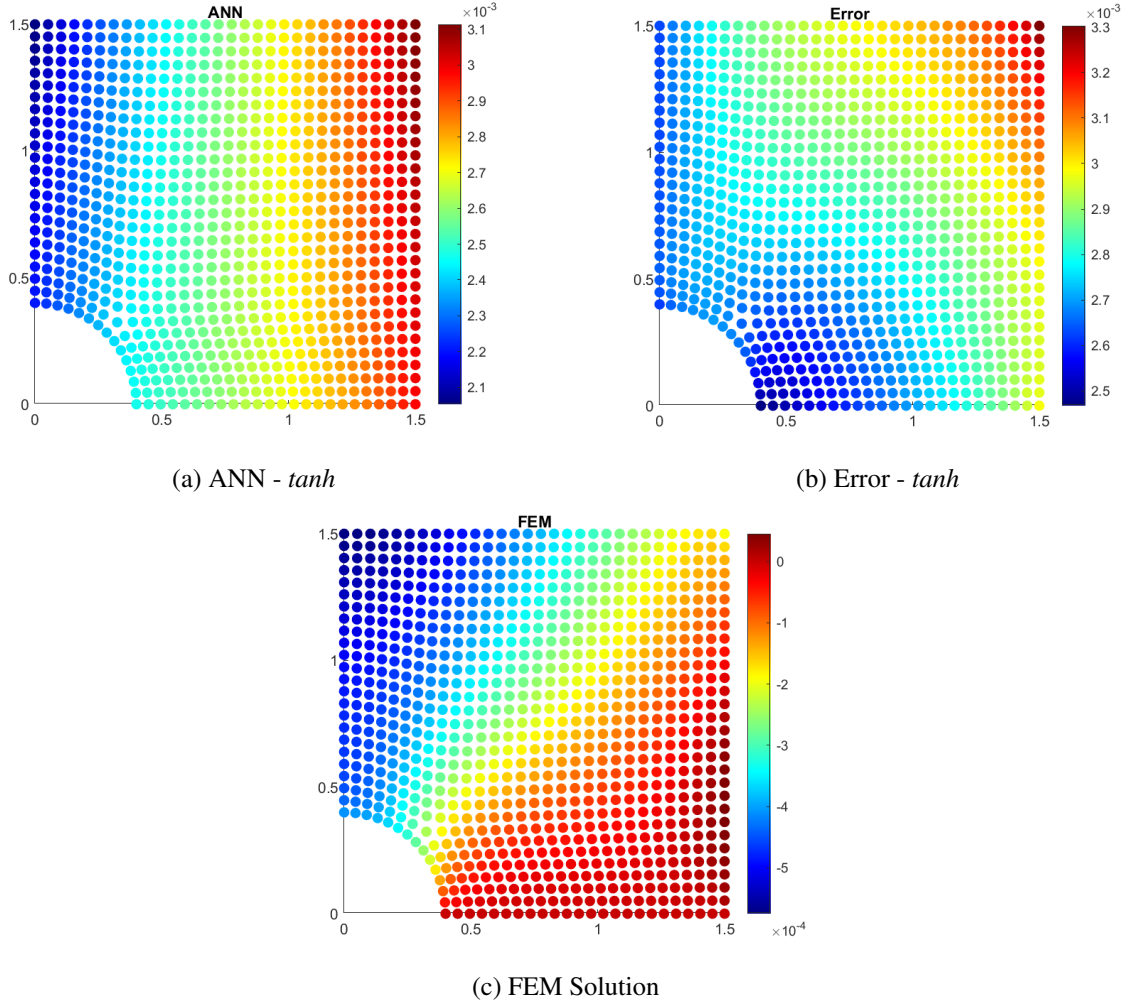


Figure 4.12: Comparison of FEM and ANN solutions for displacement v - With Outliers

The best performance of the neural network corresponds to the model using the *tanh* activation function without outliers. The optimal hyperparameters that led to the most accurate predictions and minimized the loss are:

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

4.3.2 Stress at Integration Points

The next variables to be studied are the stress at integration points, specifically in the x and y directions (σ_{xx} , σ_{yy}), and the shear stress τ_{xy} . Separate .csv files containing these data are used to train the network for each stress component individually, as this approach is anticipated to be more effective.

4.3.2.1 Integration Point - Stress σ_{xx}

The first stress component to be analysed by the neural network was the stress in the x direction, σ_{xx} . The procedure mirrors the one used for displacement, with the key difference being the use of a different file containing the corresponding stress data at the integration points. The input features remain the same, with the difference that now the coordinates of the integration points are used instead of the node coordinates.

The network is trained using the same hyperparameters and techniques as before. The training process employs the Mean Absolute Error (MAE) as the loss function, and experiments are conducted both with and without outliers to evaluate their impact on model performance.

Previous observations showed that the *ReLU* activation function was not effective in predicting displacements. Consequently, for the stress predictions, it was decided to exclusively use the *tanh* activation function.

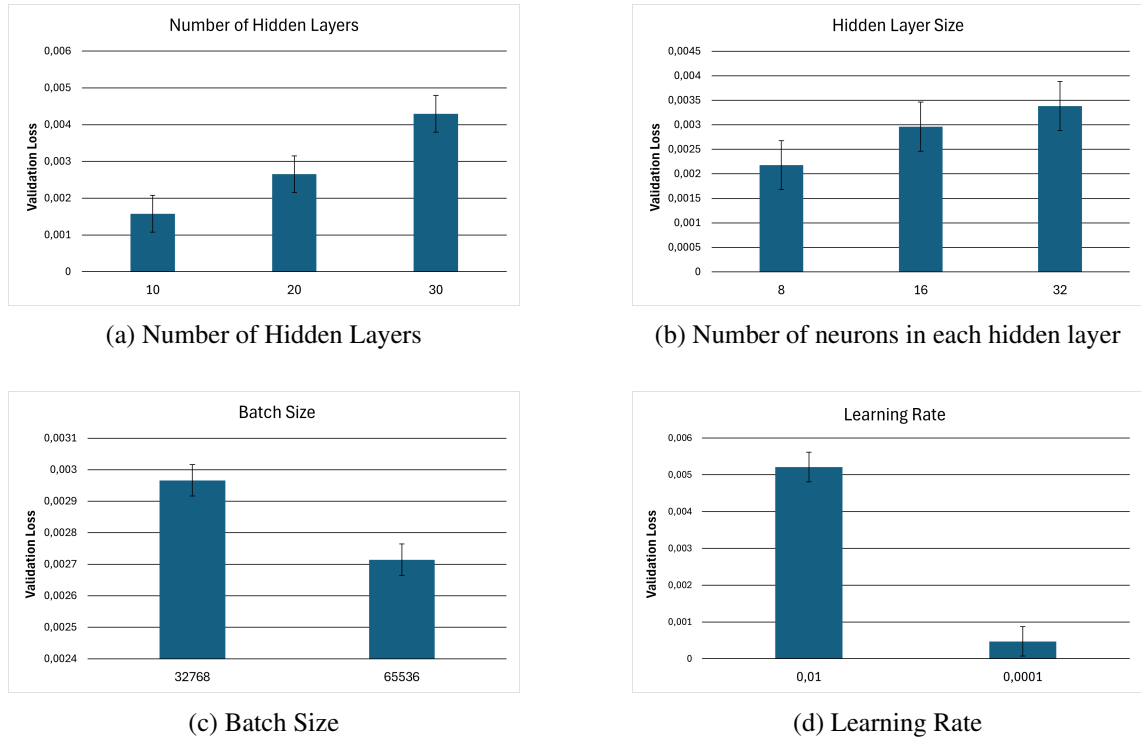


Figure 4.13: Hyperparameter sensitivity analysis for the ANN model - Excluding Outliers

The neural network trained to predict normal stress, σ_{xx} , using the *tanh* activation function and excluding outliers exhibited the results shown in Figure 4.13. The findings revealed that a smaller network configuration, consisting of 10 hidden layers with 8 neurons each, resulted in lower validation losses. Moreover, a larger batch size of 65536 outperformed a batch size of 32768. The most protuberant hyperparameter was the learning rate, as the learning rate of 0.01 resulted in an average loss that was ten times higher than that observed with a learning rate of 0.0001.

For the same network, but this time including the outliers, the results are shown in Figure 4.14. Although the validation loss appears lower, this is artificially influenced by the presence of outliers. Regarding the number of hidden layers, no definitive conclusion can be drawn as the losses are nearly identical. However, in terms of hidden layer size, a smaller layer with 8 neurons leads to better results. The most significant parameter remains the learning rate. The learning rate of 0.0001 yields much better results compared to a learning rate of 0.01.

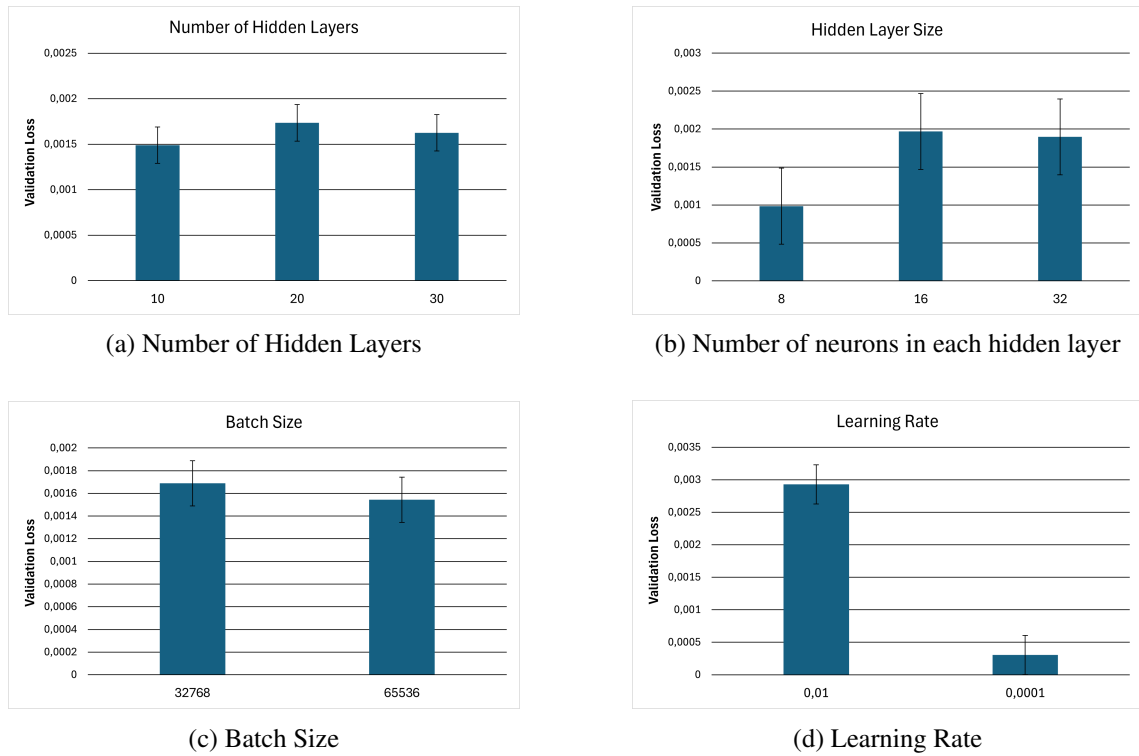
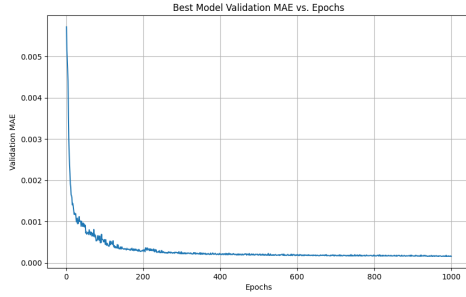


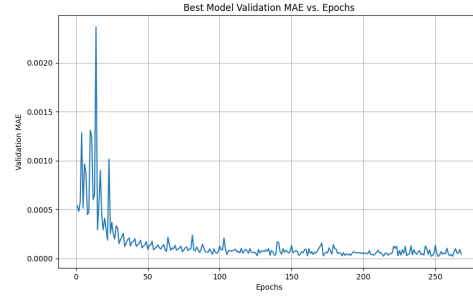
Figure 4.14: Hyperparameter sensitivity analysis for the ANN model - With Outliers

The performance of the best models, both with and without outliers, was compared to evaluate the impact of outliers on model training and validation. As it is possible to see in Figure 4.15a, the model trained without outliers exhibits a rapid decrease in validation loss within the first 200-300 epochs, after which the loss begins to converge and stabilise. Despite reaching the maximum defined limit of 1000 epochs, the loss continues to decrease, suggesting that additional epochs might further reduce the loss.

Conversely, Figure 4.15b illustrates the behavior of the model trained with outliers. During the first 50 epochs, the validation loss demonstrates significant variability. While the loss does decrease over time, it remains erratic throughout the training process. Additionally, the training process halts near 300 epochs due to the *EarlyStopping()* function, which indicates that the model had ceased improving well before reaching the 1000 epoch limit.



(a) Best Validation MAE across Epochs - Without Outliers



(b) Best Validation MAE across Epochs - With Outliers

Figure 4.15: Validation MAE across Epochs

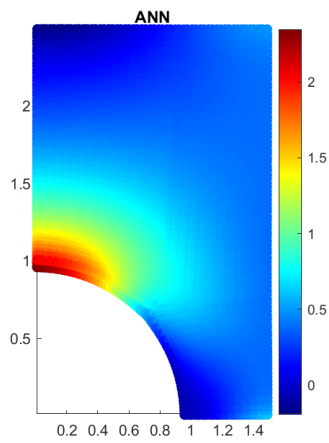
When comparing both models, it is evident that the model trained without outliers achieves a more stable and consistently lower MAE across the epochs. In contrast, the model with outliers shows more variability and a higher initial validation MAE, which, although it decreases over time, remains erratic.

The optimal hyperparameters for predicting the stress σ_{xx} , yielding the most accurate predictions and the lowest loss without outliers, are as follows:

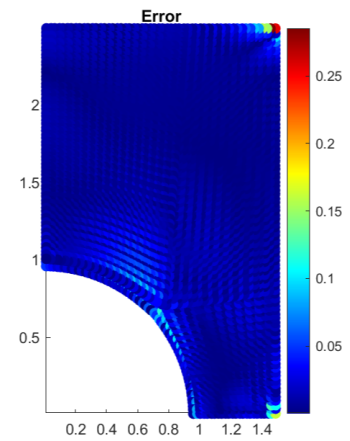
- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

To support the previous discussion on the influence of outliers, predictions on the testing data for the normal stress, σ_{xx} , were generated and presented in Figure 4.16. These predictions were made for the same length, width, and radius of the plate to provide a consistent basis for comparison.

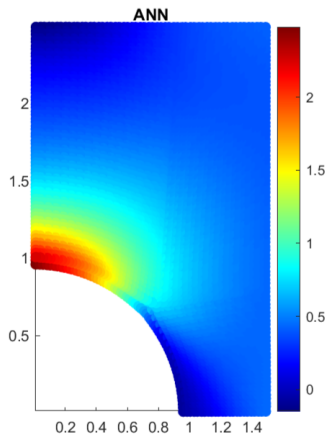
From the visualization of the Figure 4.16, it can be seen that both models, with and without outliers, perform well in predicting stresses. However, the model trained with outliers exhibits larger errors, particularly near the plate hole where stresses are highest.



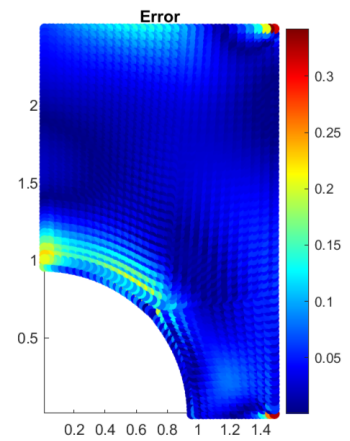
(a) ANN Prediction - Without Outliers



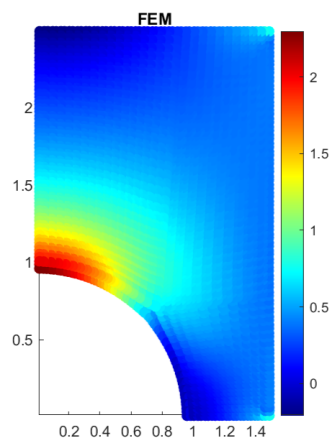
(b) Error - Without Outliers



(c) ANN Prediction - With Outliers



(d) Error - With Outliers



(e) FEM Solution

Figure 4.16: Comparison of FEM and ANN solutions for the stress - With and Without Outliers

4.3.2.2 Integration Point - Stress σ_{yy}

After analysing the stress at the integration point in the x direction, σ_{xx} , the next step was to analyse the stress in the y direction, σ_{yy} . The same procedure was applied as with σ_{xx} , using the *tanh* activation function and evaluating the influence of outliers on the data. The loss function remained the Mean Absolute Error (MAE), and the same hyperparameters as before were considered.

The neural network trained to predict the normal stress, σ_{yy} , without the presence of outliers, is shown in Figure 4.17.

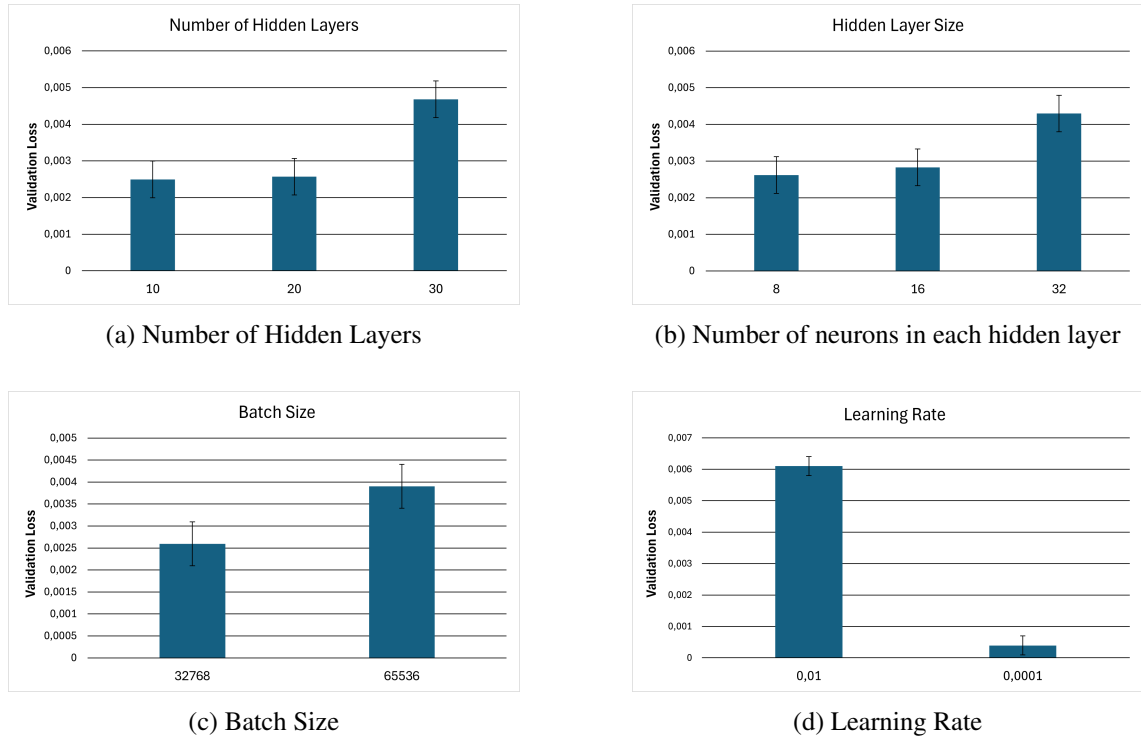


Figure 4.17: Hyperparameter sensitivity analysis for the ANN model - No Outliers

From the analysis of the data in the Figure 4.17, several insights can be extracted regarding the network without outliers. It is evident that a smaller number of hidden layers and a smaller hidden layer size result in lower loss values. Additionally, a smaller batch size of 32768 yields better results compared to larger batch sizes. Furthermore, a learning rate of 0.0001 proves to be more effective, leading to better results.

Next, the presence of outliers was considered, and the results presented in Figure 4.18.

From the analysis, it is evident that a network with 20 hidden layers produces better results. Regarding the hidden layer size, smaller sizes of 8 and 16 neurons resulted in lower losses. Additionally, a larger batch size demonstrated superior performance, and a smaller learning rate of 0.0001 significantly outperformed the higher learning rate of 0.01.

When comparing the performance of the hyperparameters for the model with and without outliers there are several things that can be extracted. In terms of the learning rate both of the

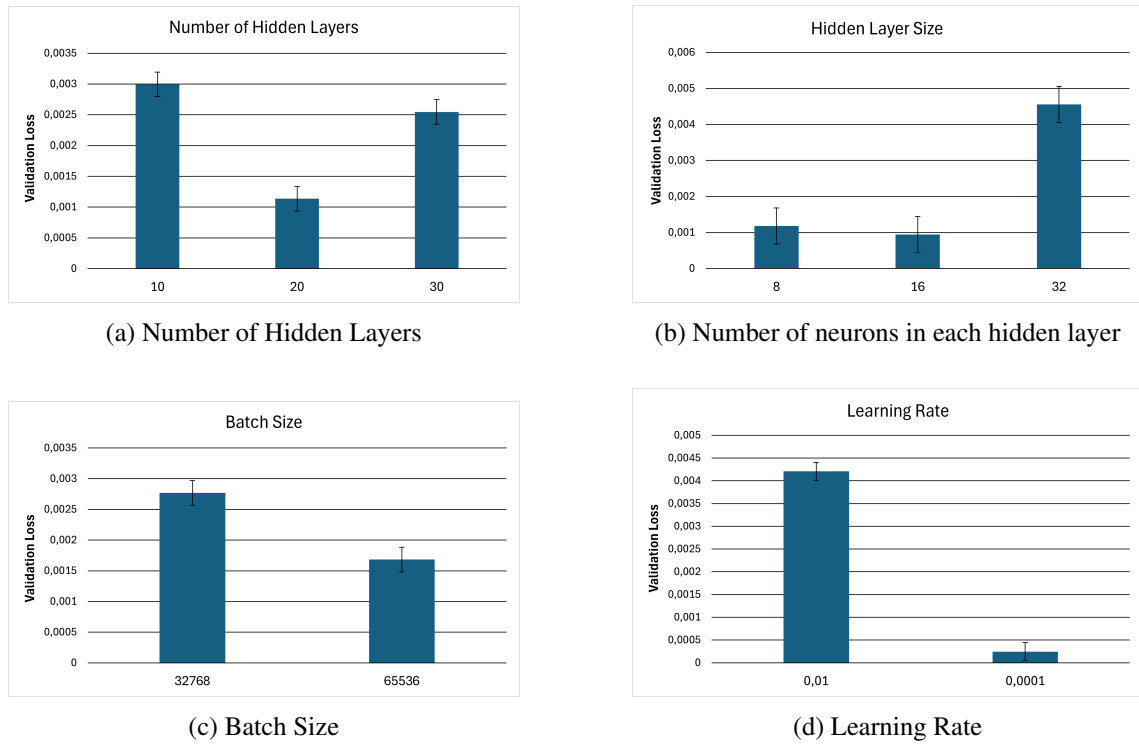


Figure 4.18: Hyperparameter sensitivity analysis for the ANN model - With Outliers

networks exhibit better results for a learning rate of 0.0001. However, the optimal batch size differs between the two models.

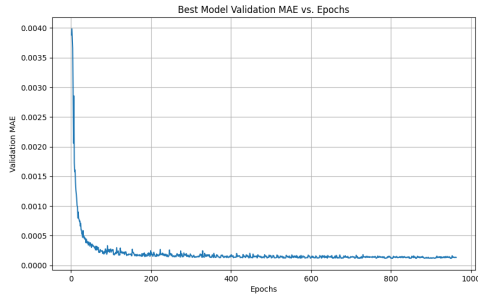
The model without outliers exhibits better results for smaller batch sizes whereas the model with outliers exhibits better results for a bigger batch size. This happens due to the fact that in the model with no outliers, which has less data, the data is more homogeneous and representative of the underlying distribution and a smaller batch size provides more frequent updates with subsets of data which leads to a better generalization in the predictions.

For the model with outliers, which has more extreme data points, a smaller batch size amplifies this effect because a smaller batch might contain a significant number of outliers, leading to more erratic updates making the process less effective. With larger batch sizes, they average the extreme values caused by outliers over more data points which reduces the impact of the outliers leading to a more stable learning.

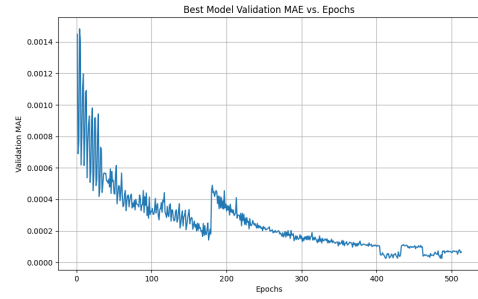
The performance of the best models, featuring the validation MAE across the epochs, represented in Figure 4.19 was compared with and without outliers, both using the *tanh* activation function.

The model without outliers exhibits an effective learning, with the validation loss decreasing rapidly within the first 200 epochs. After this initial phase, the loss converges and stabilises at a relatively low value, suggesting that the model learns well and avoids overfitting.

In contrast, the model with outliers shows high variability, mainly in the first 200 epochs. While the validation loss does decrease, it remains more erratic compared to the model without



(a) Best Validation MAE across Epochs - Without Outliers



(b) Best Validation MAE across Epochs - With Outliers

Figure 4.19: Validation MAE across Epochs

outliers. The presence of outliers causes the model to struggle with a stable learning, leading to erratic validation loss and an earlier stopping point.

The predictions of the normal stress σ_{yy} for both models, with and without outliers, were made and are presented for a plate with random dimensions in Figure 4.20. The model with outliers demonstrates more difficulty in predicting the stress σ_{yy} compared to the model without outliers, as evidenced by the error plot in Figure 4.20d.

The best performance of the network is achieved using the *tanh* activation function with no outliers in the data. The optimal set of hyperparameters for this configuration is given by:

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 65536
- **Learning Rate:** 0.0001

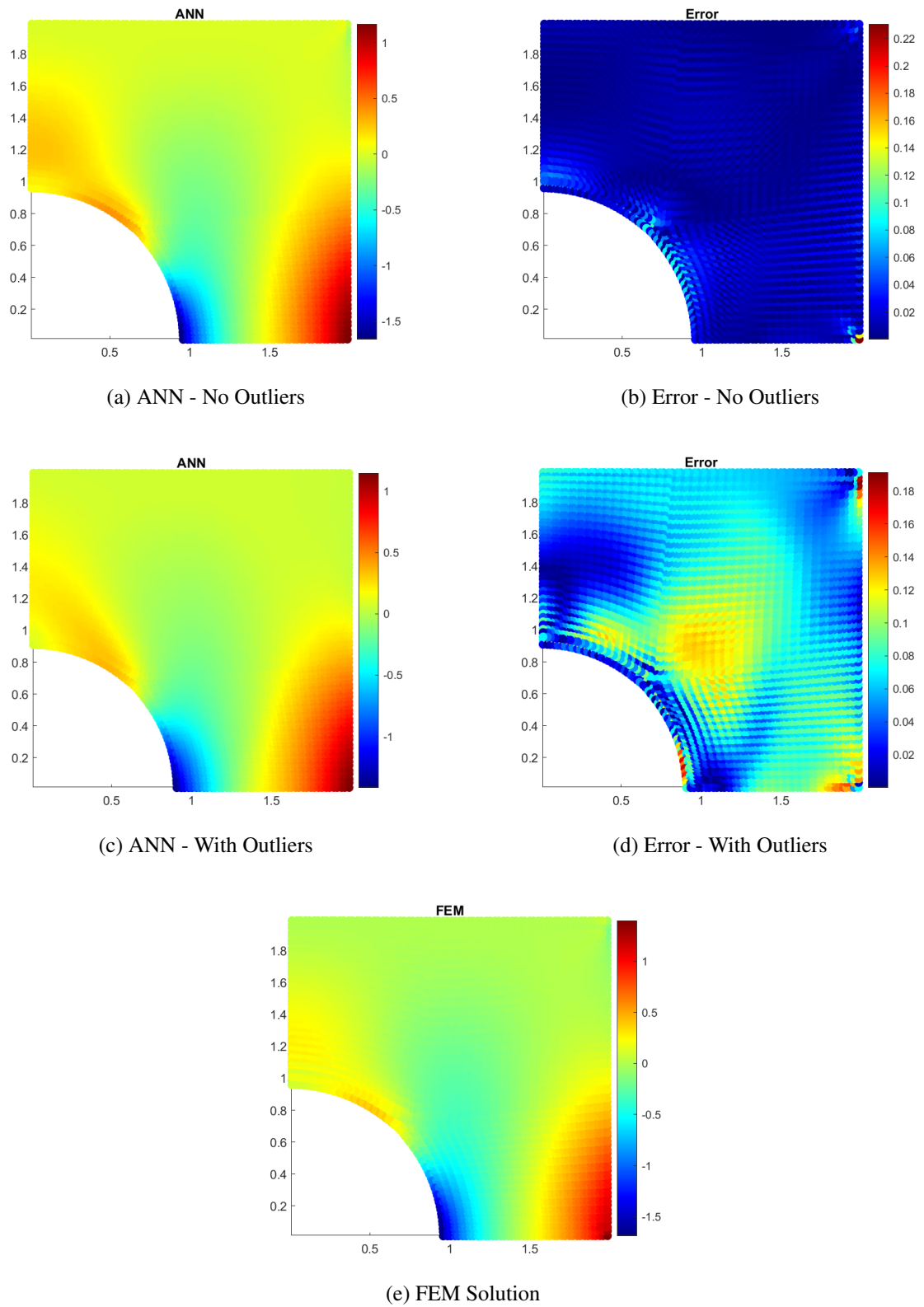


Figure 4.20: Comparison of FEM and ANN solutions for the stress σ_{yy} - With and Without Outliers

4.3.2.3 Integration Point - Stress τ_{xy}

The next variable studied was the shear stress component, τ_{xy} , in the xy plane. The same procedure as before was applied, using the *tanh* activation function and evaluating the influence of outliers on the data with the Mean Absolute Error (MAE) as the loss function.

First, the network was trained without outliers. The evaluation of the hyperparameters in predicting the stress τ_{xy} without outliers is presented in Figure 4.21.

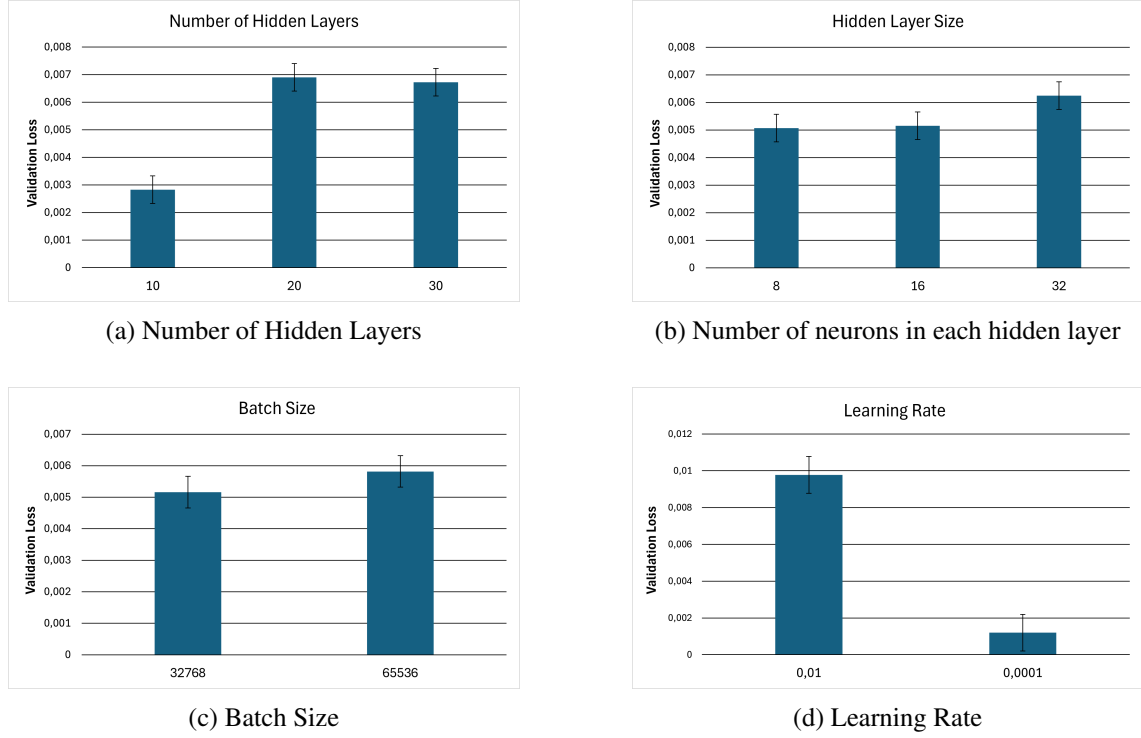


Figure 4.21: Hyperparameter sensitivity analysis for the ANN model - Without Outliers

As evident from the Figure 4.21, a smaller number of hidden layers (10) achieves significantly better results compared to larger configurations. Additionally, a learning rate of 0.0001 results in a much smaller validation loss than a learning rate of 0.01.

The presence of outliers was also considered, and the results presented in Figure 4.22.

From this analysis, it is evident that a smaller number of hidden layers yields better results. Additionally, a smaller hidden layer size and a larger batch size result in lower validation loss. Most notably, a learning rate of 0.0001 produces significantly better outcomes compared to a learning rate of 0.01.

The performance of the model, both with and without outliers, is illustrated by the best validation MAE for each of the best model (with and without outliers) across the epochs, as shown in the following Figure 4.23.

After analysing the performance of both models, with and without outliers, it is evident that they both exhibit effective and stable learning. The validation loss decreases rapidly during the

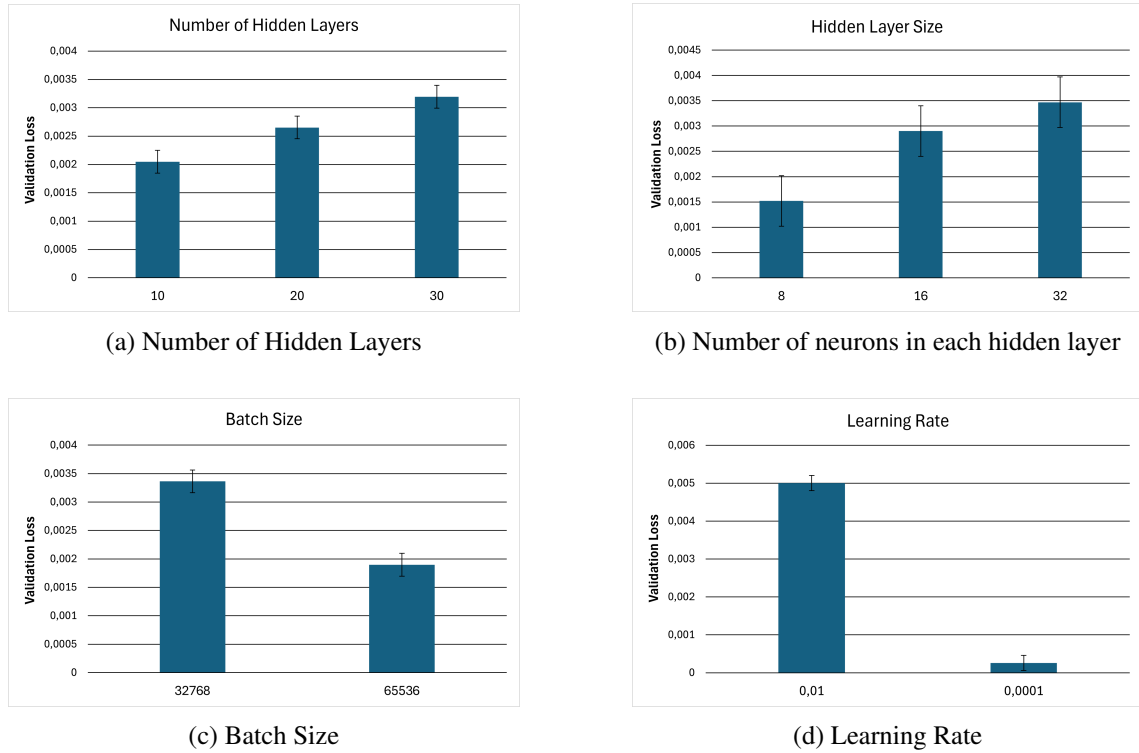


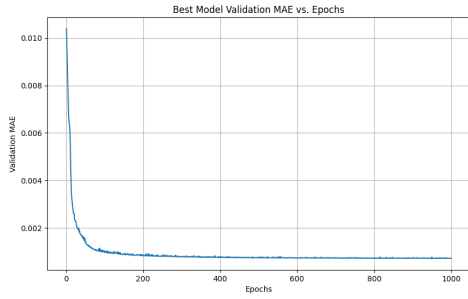
Figure 4.22: Hyperparameter sensitivity analysis for the ANN model - With Outliers

first 200 epochs before stabilising. The primary difference between the models is that the one with outliers shows a small degree of variability during the first 50 epochs, though this variability is not significant.

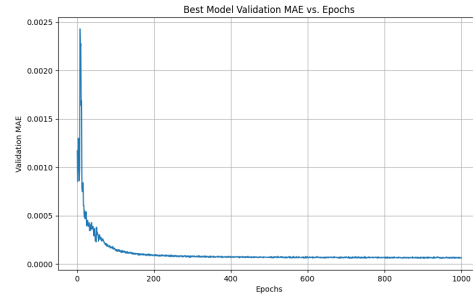
For the normal stresses, σ_{xx} and σ_{yy} , the variability on the model with outliers was clearly visible, as well as, the difficulty to maintain a stable learning. However, when predicting the shear stress, τ_{xy} , the model with outliers performed similarly to the model without outliers, with the MAE across epochs showing nearly identical learning curves. This happens because normal stresses are more concentrated near the hole, where the outliers are more prevalent. On the other hand, the shear stress is more evenly distributed which reduces the impact of localized outliers.

The predictions for the shear stress τ_{xy} for the best models with and without outliers are presented in Figure 4.24. The predictions of the neural network were made for a plate with 2 x 1.5 dimensions with a hole radius of 0.55, and were compared to the FEM solution.

From the visualization of the figures both of the models perform very similarly. The error margins for the neural network predictions are identical in distribution nevertheless, the error is higher with outliers.



(a) Best Validation MAE across Epochs - Without Outliers



(b) Best Validation MAE across Epochs - With Outliers

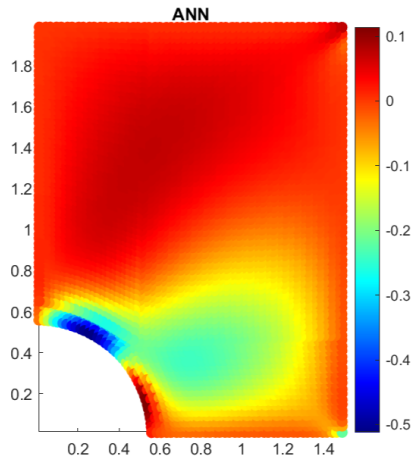
Figure 4.23: Validation MAE across Epochs

The best set of hyperparameters for predicting the shear stress τ_{xy} using the *tanh* activation function, without outliers in the data, is as follows:

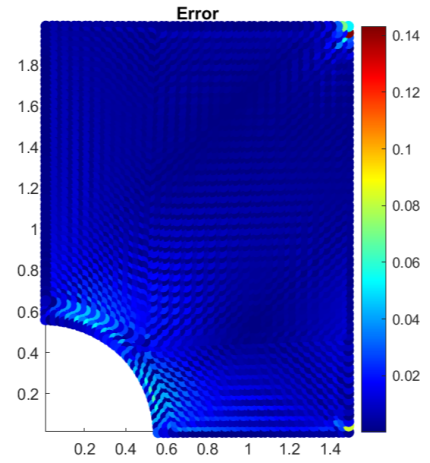
- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

For the model with outliers and using the same activation function, the optimal hyperparameters are:

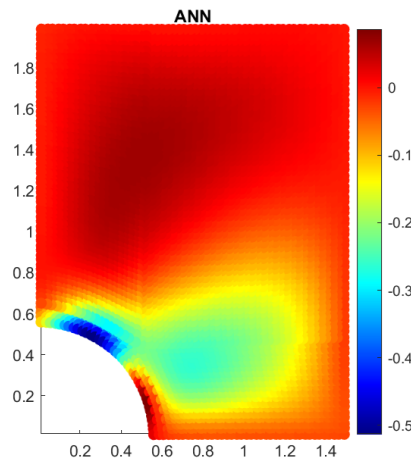
- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001



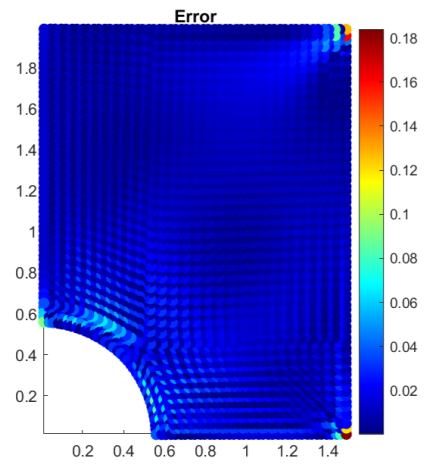
(a) ANN - No Outliers



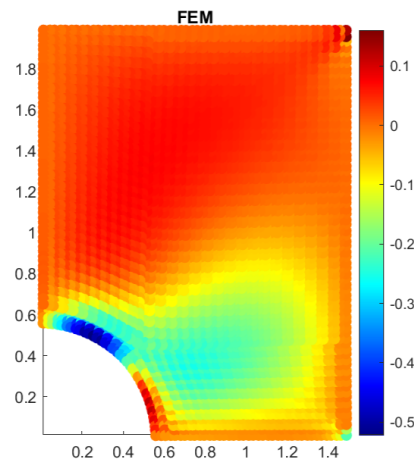
(b) Error - No Outliers



(c) ANN - With Outliers



(d) Error - With Outliers



(e) FEM Solution

Figure 4.24: Comparison of FEM and ANN for the shear stress τ_{xy} - With and Without Outliers

4.3.3 Strain at Integration Points

The next outputs to be examined are the strains at the integration points. These include the strain in the x direction (ϵ_{xx}), the strain in the y direction (ϵ_{yy}), and the shear strain in the xy plane (ϵ_{xy}).

4.3.3.1 Integration Point - Strain ϵ_{xx}

The first strain component to be analysed at the integration point is the strain component in the x direction, ϵ_{xx} . The input features to train the network remain the same with those used for predicting stress at the integration points, and the same training procedure is followed.

The network is trained using the same set of hyperparameters and the Mean Absolute Error (MAE) as the loss function. The neural network is trained with and without outliers on the data in order to evaluate their impact on the model performance. Additionally, the $\tanh$ function will be used as the activation function.

The results of the neural network without the presence of outliers is shown in Figure 4.25.

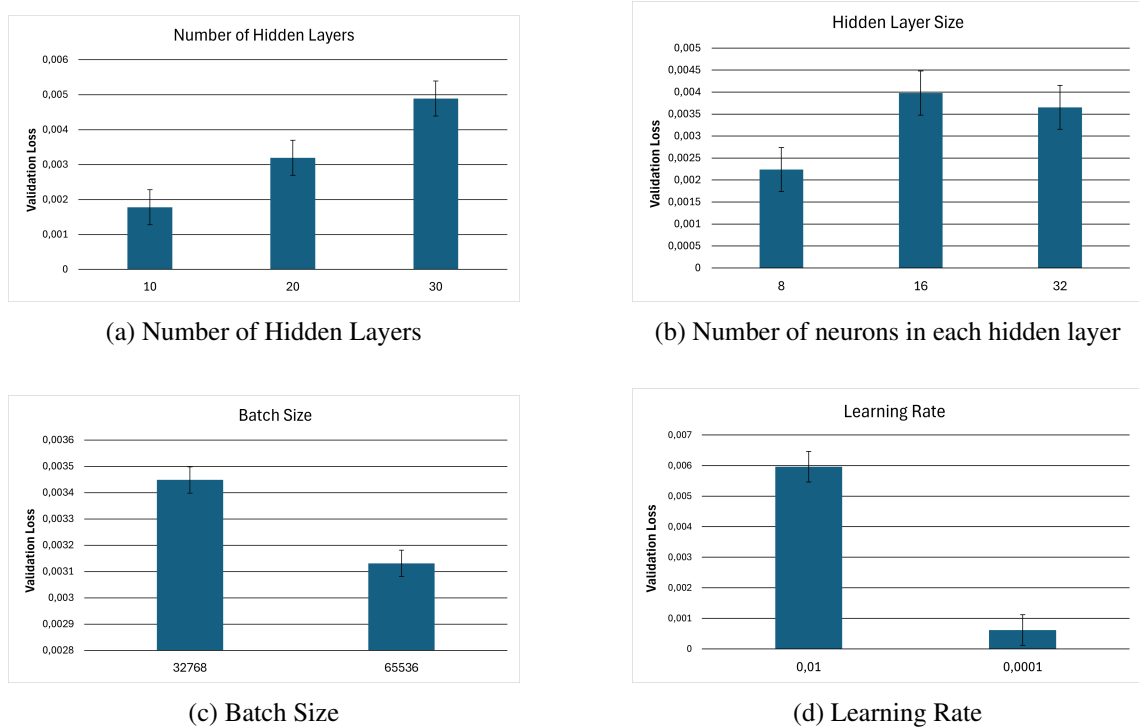


Figure 4.25: Hyperparameter sensitivity analysis for the ANN model - Without Outliers

From the figures, several insights can be drawn regarding the impact of hyperparameters on validation loss. A smaller number of hidden layers, specifically 10, results in a lower validation loss compared to configurations with more layers. Similarly, a smaller number of neurons in each hidden layer yields better results.

For the batch size, a bigger batch of 65536 leads to a much smaller validation loss and in terms of the learning rate, the learning rate of 0.0001 has achieved better results when comparing to the learning rate of 0.01.

The presence of outliers on the data was also considered, and the results presented in Figure 4.26.

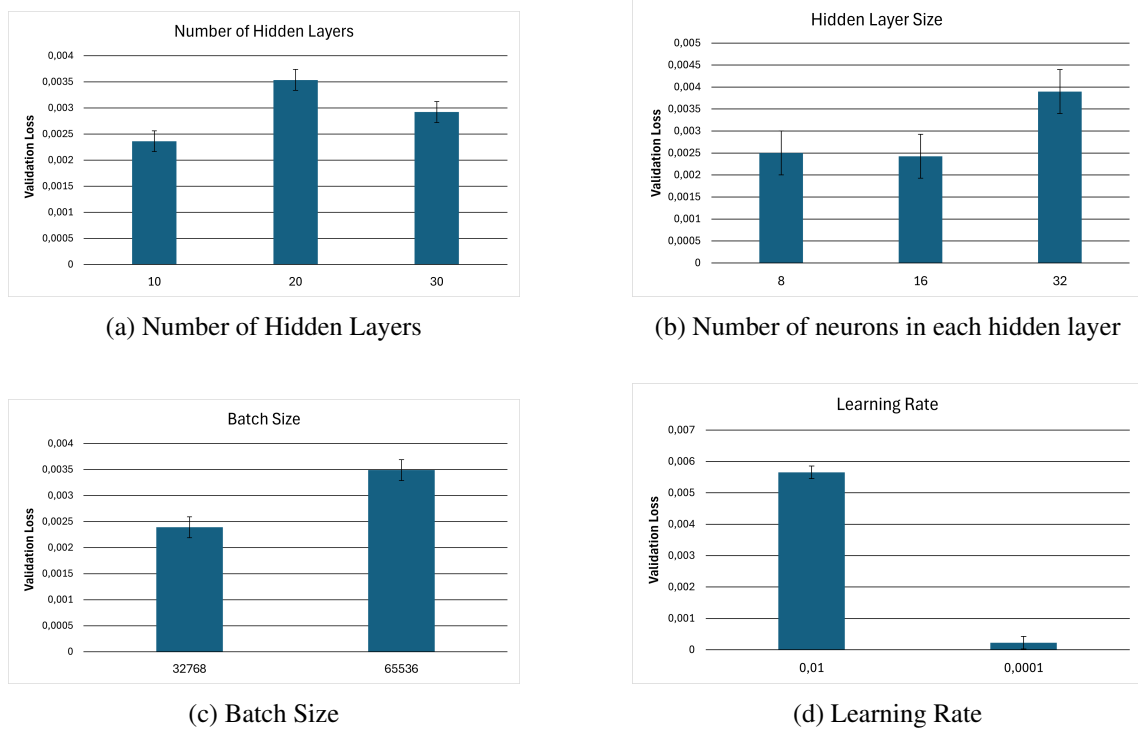


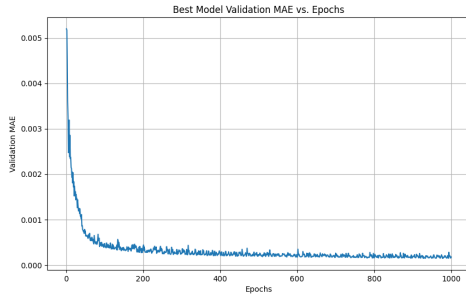
Figure 4.26: Hyperparameter sensitivity analysis for the ANN model - With Outliers

The results differ slightly when comparing the model with outliers to the one without. For both models, a smaller learning rate of 0.0001 consistently yields better results than a learning rate of 0.01. However, in the presence of outliers, a smaller batch size of 32768 proves to be more effective. Additionally, a smaller number of hidden layers leads to a lower validation loss, and hidden layer sizes of 8 and 16 neurons produce better results.

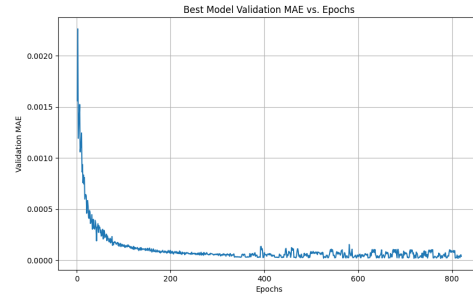
The training performance of the best models, both with and without the presence of outliers, is represented in Figure 4.27 which shows the validation MAE for each model over the course of the epochs.

For the best model without the presence of outliers, the validation MAE starts decreasing more sharply in the first 200 epochs and begins to converge at around 400 epochs. This indicates that the model learns effectively and exhibits stable learning.

The best model with outliers also shows a sharp decrease in validation MAE within the first 200 epochs but struggles to stabilize and converge. It displays slight variability in the later epochs. Consequently, the training process is halted by the *EarlyStopping()* function near 800 epochs due to these fluctuations.



(a) Best Validation MAE across Epochs - Without Outliers



(b) Best Validation MAE across Epochs - With Outliers

Figure 4.27: Validation MAE across Epochs

The best set of hyperparameters for the model **without outliers** is as follows:

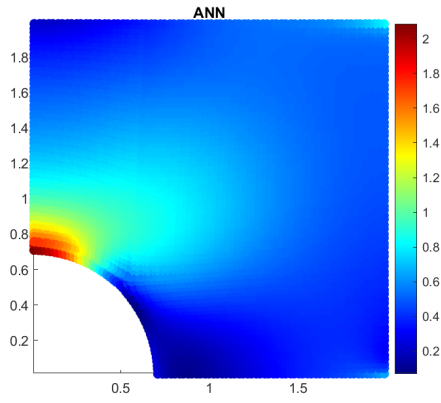
- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

For the model **with outliers**, the optimal hyperparameters are adjusted to accommodate the variability introduced by the outliers:

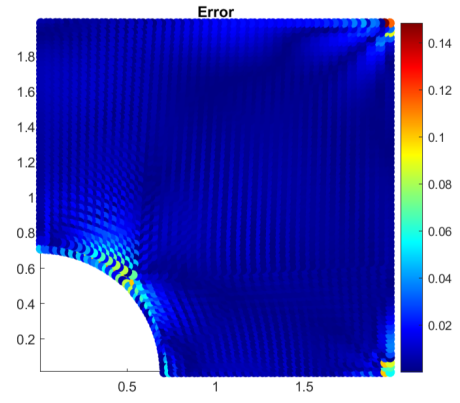
- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 65536
- **Learning Rate:** 0.0001

The predictions for the strain in the x direction, ϵ_{xx} , both with and without outliers, are illustrated in Figure 4.28 and were conducted for plate with dimensions of 2×2 .

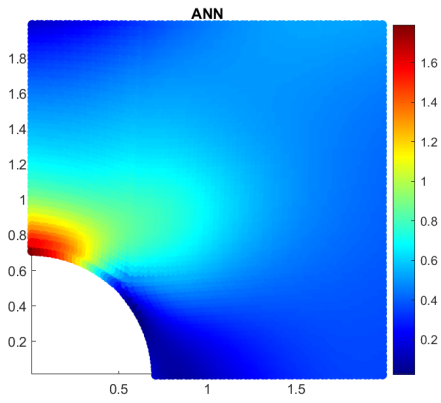
From the Figure 4.28, it is evident that the neural network can achieve accurate predictions both with and without the presence of outliers in the data. However, the model with outliers exhibits a higher error, reinforcing the earlier observations regarding the impact of outliers on model performance.



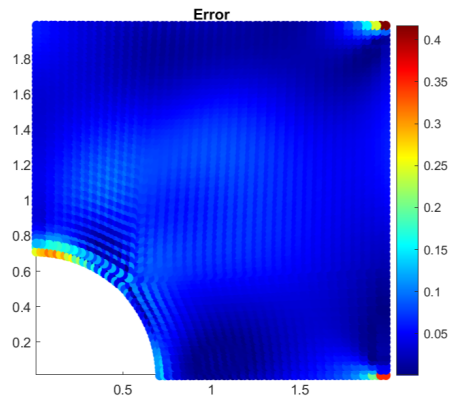
(a) ANN - No Outliers



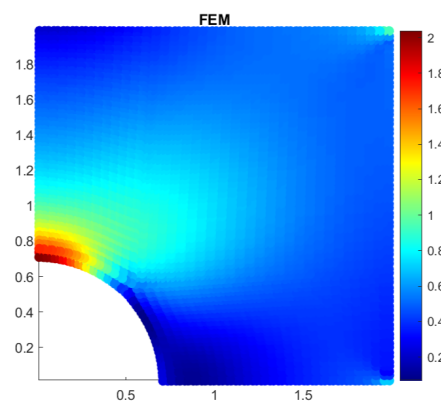
(b) Error - No Outliers



(c) ANN - With Outliers



(d) Error - With Outliers



(e) FEM Solution

Figure 4.28: Comparison of FEM and ANN for the strain ϵ_{xx} - With and Without Outliers

4.3.3.2 Integration Point - Strain ε_{yy}

The next output to be analysed is the strain at the integration point in the y direction, ε_{yy} . As before, the impact of outliers on the model's performance and accuracy will be examined. The *tanh* activation function will be used for this purpose.

Initially, the network was trained without the presence of outliers. The evaluation of the hyperparameters in predicting the strain ε_{yy} is presented in Figure 4.29.

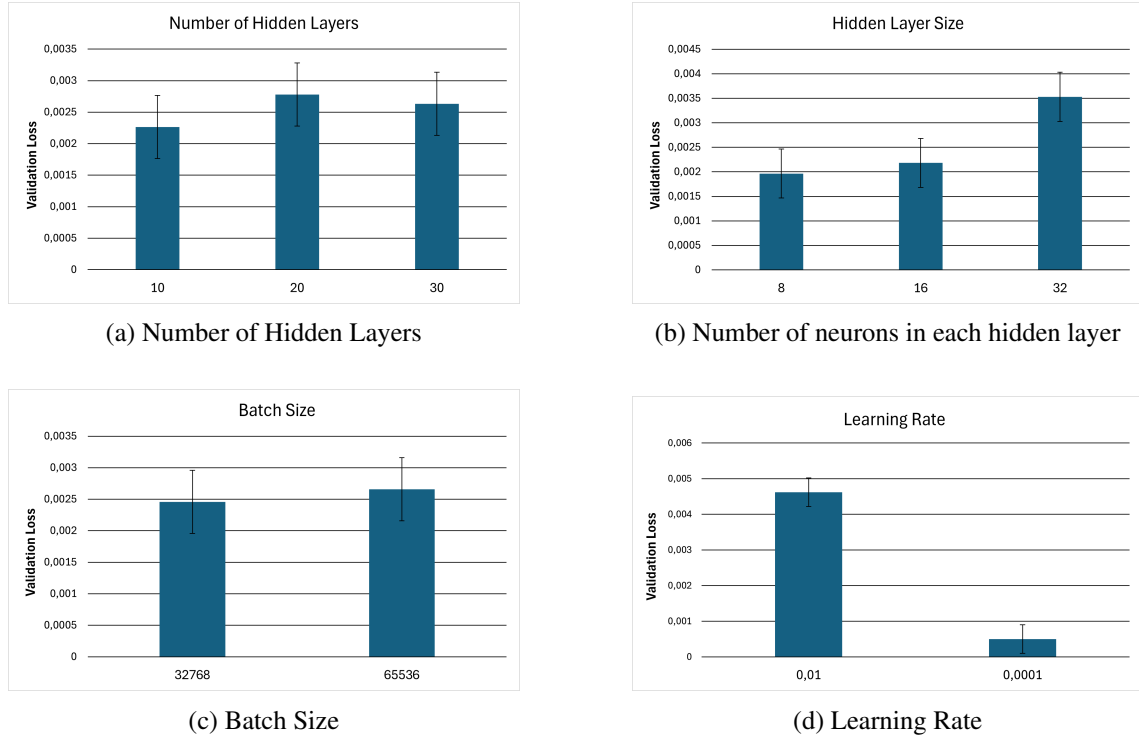


Figure 4.29: Hyperparameter sensitivity analysis for the ANN model - Without Outliers

From the Figure 4.29, it can be observed that the validation loss remains nearly constant across different numbers of hidden layers and batch sizes. However, smaller hidden layer sizes yield better results and, when comparing learning rates, the smaller learning rate of 0.0001 significantly outperforms the learning rate of 0.01.

Next, the impact of outliers on the model's performance was considered, as shown in Figure 4.30.

The Figure 4.30 indicate that a smaller hidden layer size of 8 neurons per layer generally leads to lower validation losses. When considering batch size, the largest batch size yields the best performance with the lowest validation loss. Finally, a learning rate of 0.0001 consistently outperforms a learning rate of 0.01.

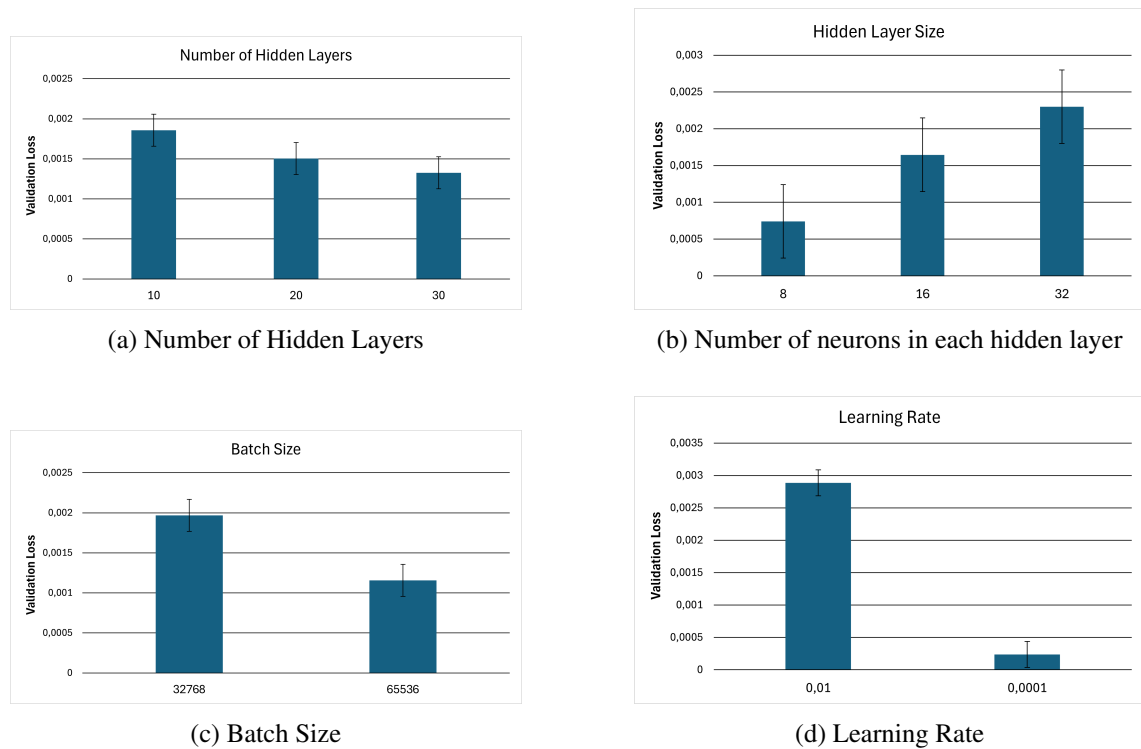


Figure 4.30: Hyperparameter sensitivity analysis for the ANN model - With Outliers

The training performance of the best models, with and without the presence of outliers, can be seen in Figure 4.31 where the validation MAE along the epochs is represented.

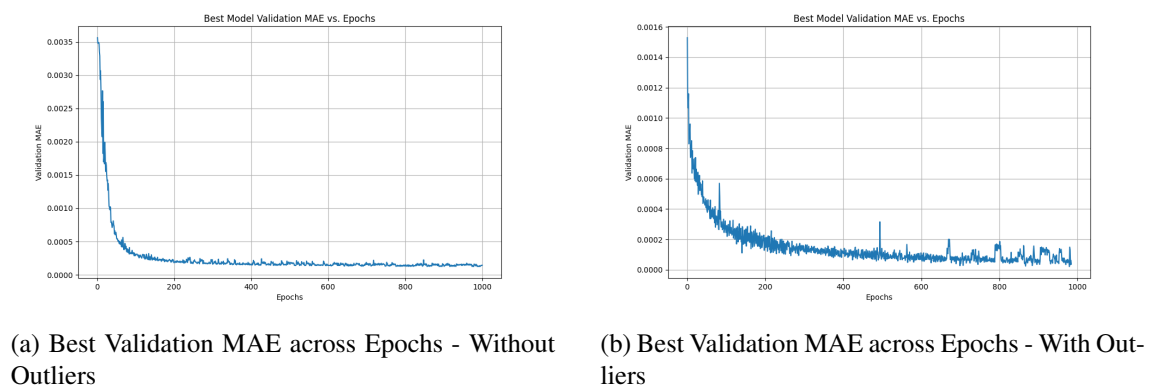


Figure 4.31: Validation MAE across Epochs

The comparison of both of Figures 4.31a and 4.31b highlights the influence of outliers on the model's training stability and accuracy. In Figure 4.31a, the model demonstrates a rapid decrease within the first 200 epochs. After this initial phase the validation MAE stabilizes and converges, indicating a steady learning process with minimal fluctuations.

In figure 4.31b, the validation MAE also decreases sharply during the first 200 epochs however, the model exhibits more variability and takes longer to stabilize. This fluctuation continues, albeit

less pronounced, up to around 800 epochs.

Figure 4.31 clearly illustrate that while the neural network is capable of achieving good predictions with and without outliers, the presence of outliers introduces higher error rates and instability in the model's training process.

The best set of hyperparameters for the model **without outliers** are as follows,

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

Whereas the optimal hyperparameters for the model **with outliers** are:

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

Comparing the best set of hyperparameters, the primary difference lies in the number of hidden layers. The model without outliers is configured with 10 hidden layers, whereas the model with outliers has 30 hidden layers.

The predictions, made on the testing data, were made in order to illustrate the accuracy of the networks in predicting the ϵ_{yy} strain. These same predictions are made for a plate with dimensions of 2 x 1.5 with a hole of 0.65 mm and can be seen in Figure 4.32.

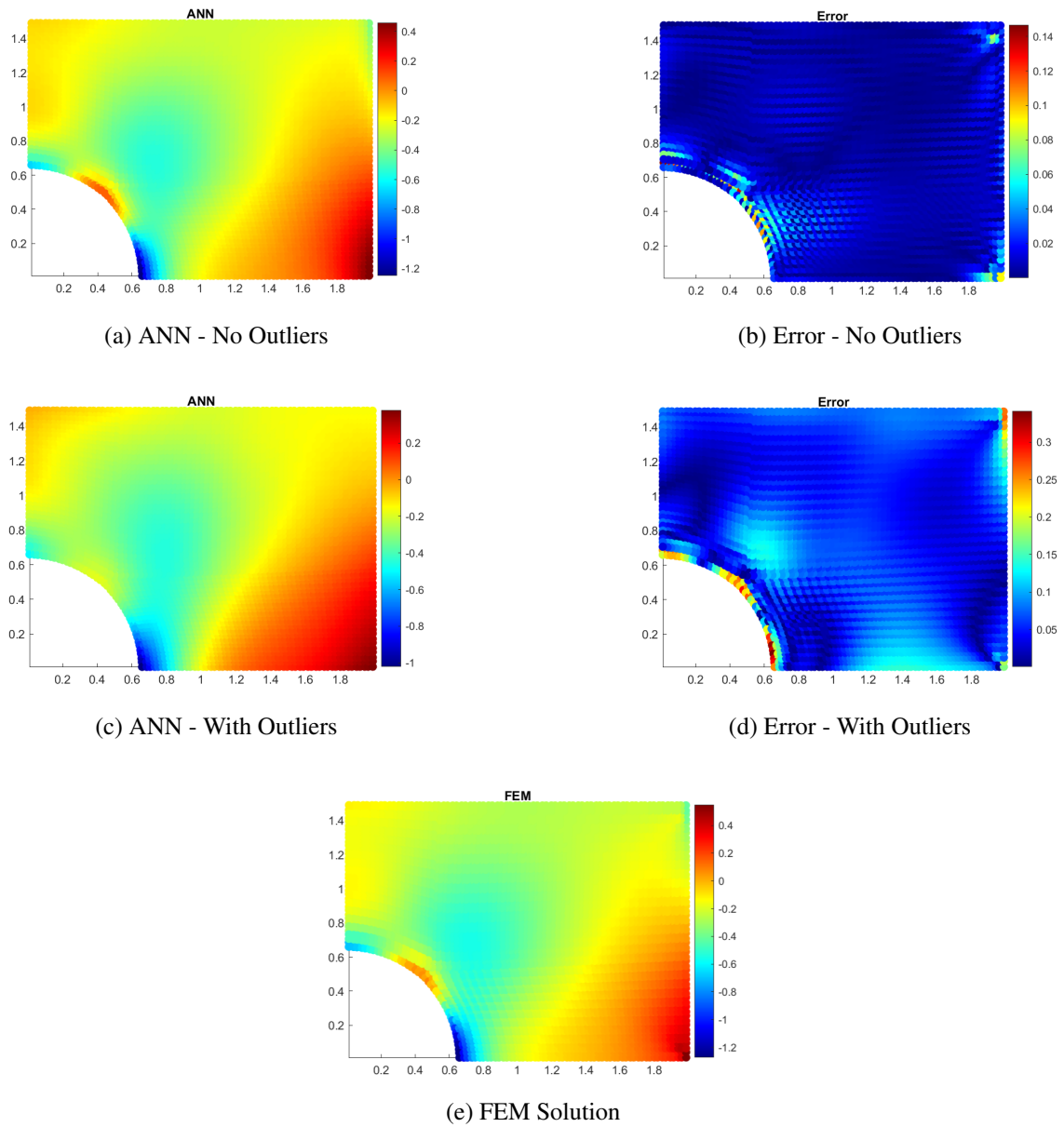


Figure 4.32: Comparison of FEM and ANN for the strain ϵ_{yy} - With and Without Outliers

From the Figure 4.32 above it can be seen that the model with no outliers has more difficulty in predicting the strain in the y direction. That difficulty is mainly localized near the plate hole, where the absolute error is bigger. Despite this, the model manages to almost correctly pick up the patterns of the ϵ_{yy} strain as in the FEM solution. The model with no outliers presents a better accuracy with a minor error, proving what has previously said on the performance of the model without outliers.

4.3.3.3 Integration Point - Strain ϵ_{xy}

The next output to be studied for the integration point was the shear strain ϵ_{xy} , in the xy plane. The same procedure as before was applied with the *tanh* as the activation function, and evaluating the influence of outliers on the model performance.

The influence of the hyperparameters on the shear strain ϵ_{xy} without the presence of outliers is presented in the following Figure 4.33.

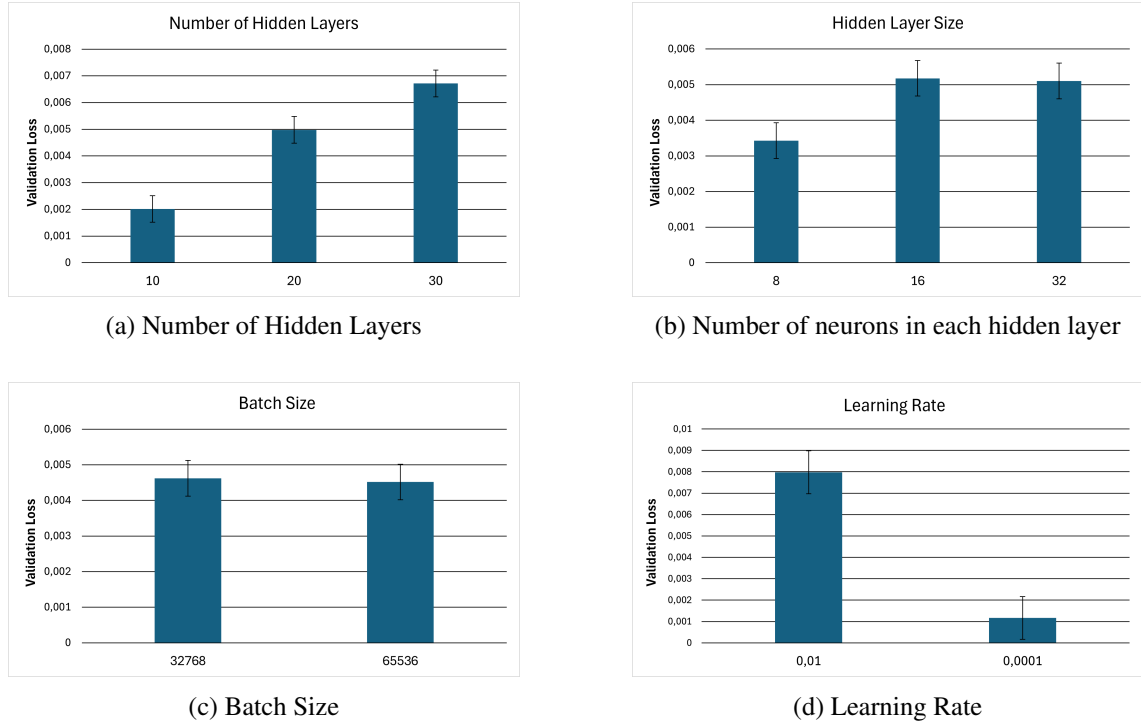


Figure 4.33: Hyperparameter sensitivity analysis for the ANN model - Without Outliers

From the visual analysis of the Figure 4.33, it can be concluded that a smaller number of hidden layers, specifically 10, results in a lower validation loss. Similarly, a hidden layer size of 8 neurons also contributes to a reduced validation loss. The learning rate of 0.0001 significantly outperforms the learning rate of 0.01, leading to much better overall results.

The influence of outliers on the model's performance was considered and the results presented in Figure 4.34.

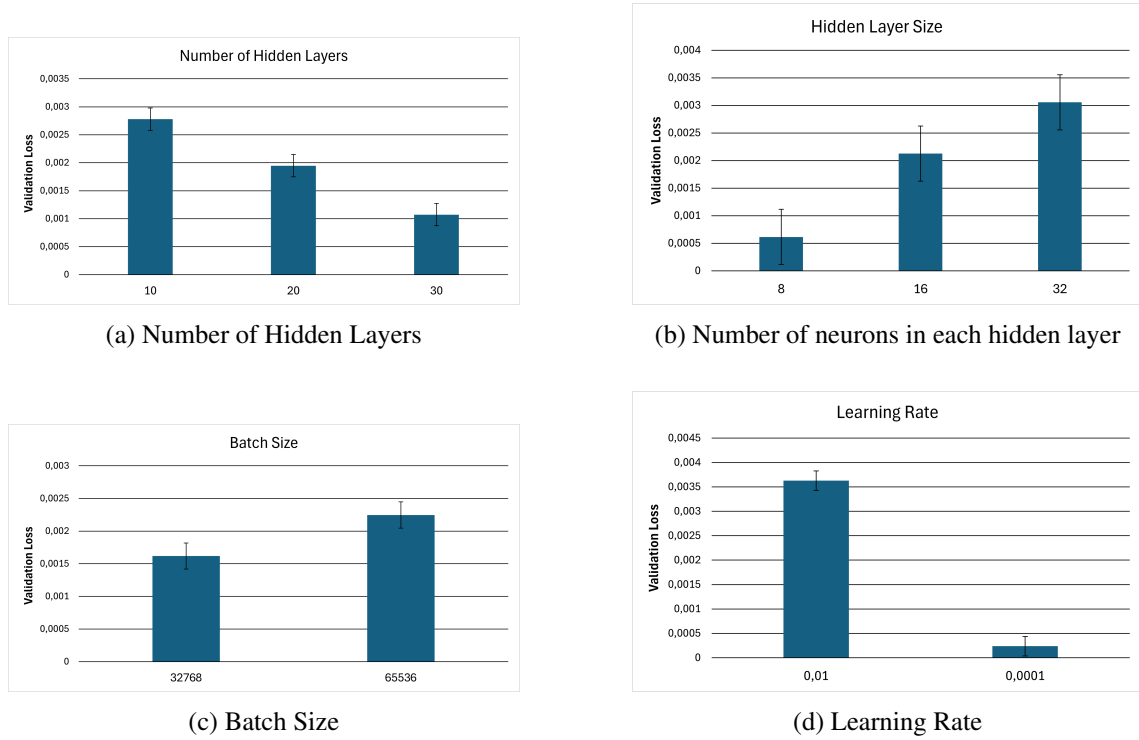


Figure 4.34: Hyperparameter sensitivity analysis for the ANN model - With Outliers

From the Figure 4.34, corresponding to the presence of outliers, several conclusions can be drawn:

1. **Number of Hidden Layers:** Increasing the number of hidden layers to 30 results in a lower validation loss, indicating better model performance.
2. **Hidden Layer Size:** A smaller hidden layer size of 8 neurons per layer leads to improved results, as evidenced by the lower validation loss.
3. **Batch Size:** A smaller batch size of 32768 is more effective, resulting in better model performance compared to the larger batch size.
4. **Learning Rate:** A learning rate of 0.0001 significantly outperforms the learning rate of 0.01.

The performance of the best models, both with and without outliers, will be evaluated by assessing the validation MAE across epochs. This comparison is illustrated in Figure 4.35.

From the comparison of the Figure 4.35, it is evident that both models, with and without outliers, perform exceptionally well in predicting the shear strain ϵ_{xy} . The validation MAE for both models decreases rapidly during the first 200 epochs before stabilizing at a very low value. Furthermore, both models exhibit stable and effective learning, indicating strong performance and accurate predictions.

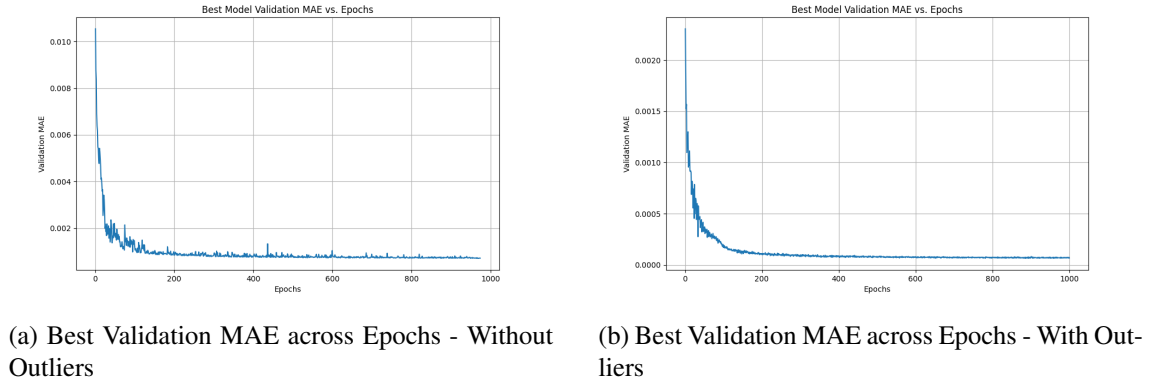


Figure 4.35: Validation MAE across Epochs

For the best model **without outliers**, the best set of hyperparameters is given by:

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

For the model **with outliers**, the optimal hyperparameters are as follows:

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 65536
- **Learning Rate:** 0.0001

The key difference between the optimal hyperparameters for the two models lies in the number of hidden layers and the batch size.

The predictions on the testing data for the best models, both with and without outliers, were made for the shear strain ϵ_{xy} . These predictions were conducted on a plate with dimensions of 1.5 x 3 with a hole radius of 1.1 mm and are represented in Figure 4.36.

From the Figure 4.36 showing the predictions, several conclusions can be drawn about the models' performance. Both the model with outliers and the model without outliers perform exceptionally well in predicting the strain ϵ_{xy} . This confirms the earlier observations regarding the models' performance across epochs in Figure 4.35, where both models demonstrated a stable learning.

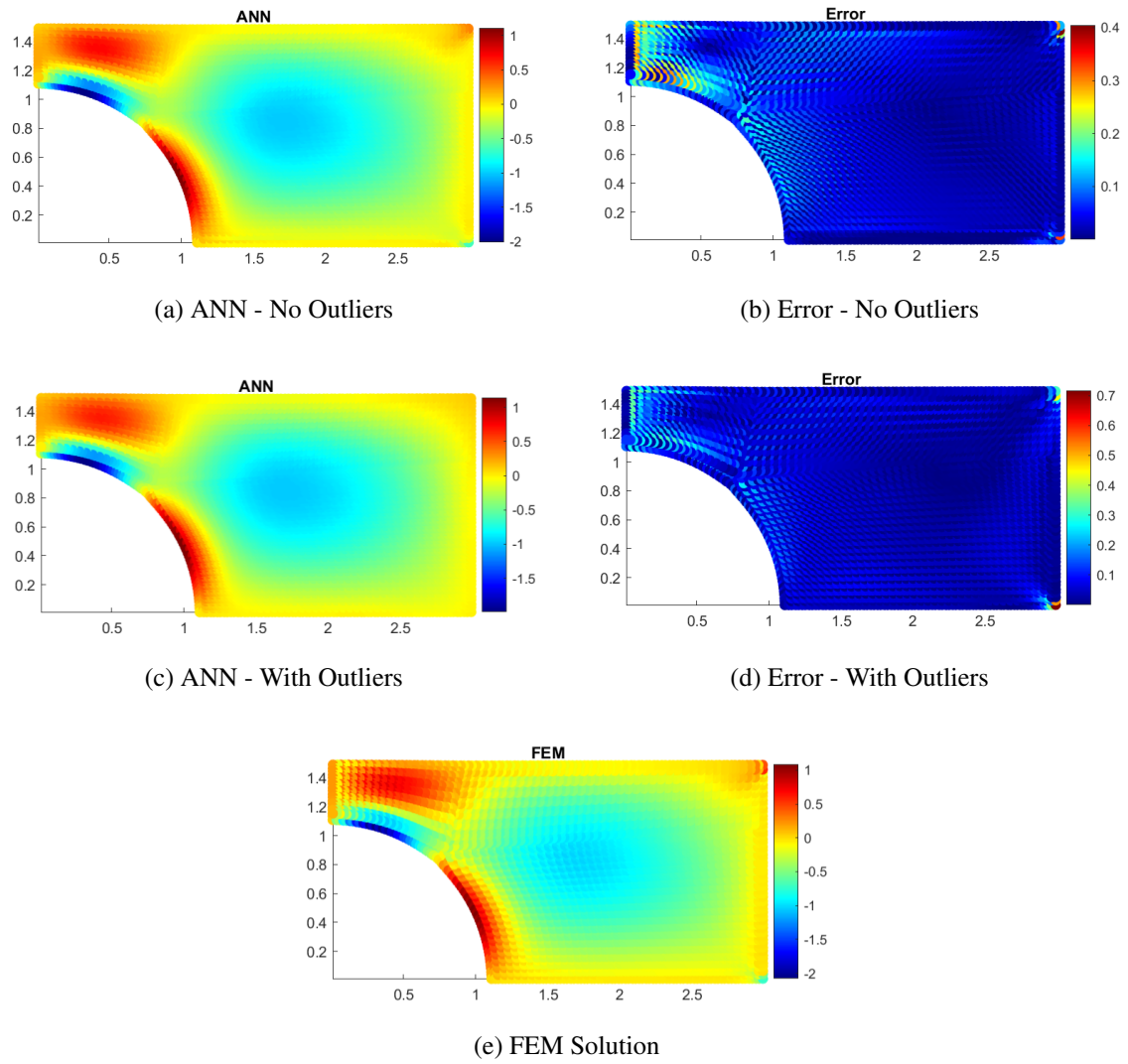


Figure 4.36: Comparison of FEM and ANN for the shear strain ϵ_{xy} - With and Without Outliers

4.3.4 Stress at Nodes

The neural network, originally trained for predicting the outputs for the integration points, was used to predict the outputs for the nodes. The goal of this approach is to assess whether the model, trained on integration points, could effectively predict stress values at the nodes as well. The code included in Annex A.6 pertains to the predictions at the nodes for the stress component σ_{xx} .

4.3.4.1 Stress σ_{xx}

The predictions made for the stress in the x direction can be seen in the following Figure 4.37. From the comparison of the figures, it is possible to extract that the neural network prediction aligns well with the actual stress value, σ_{xx} .

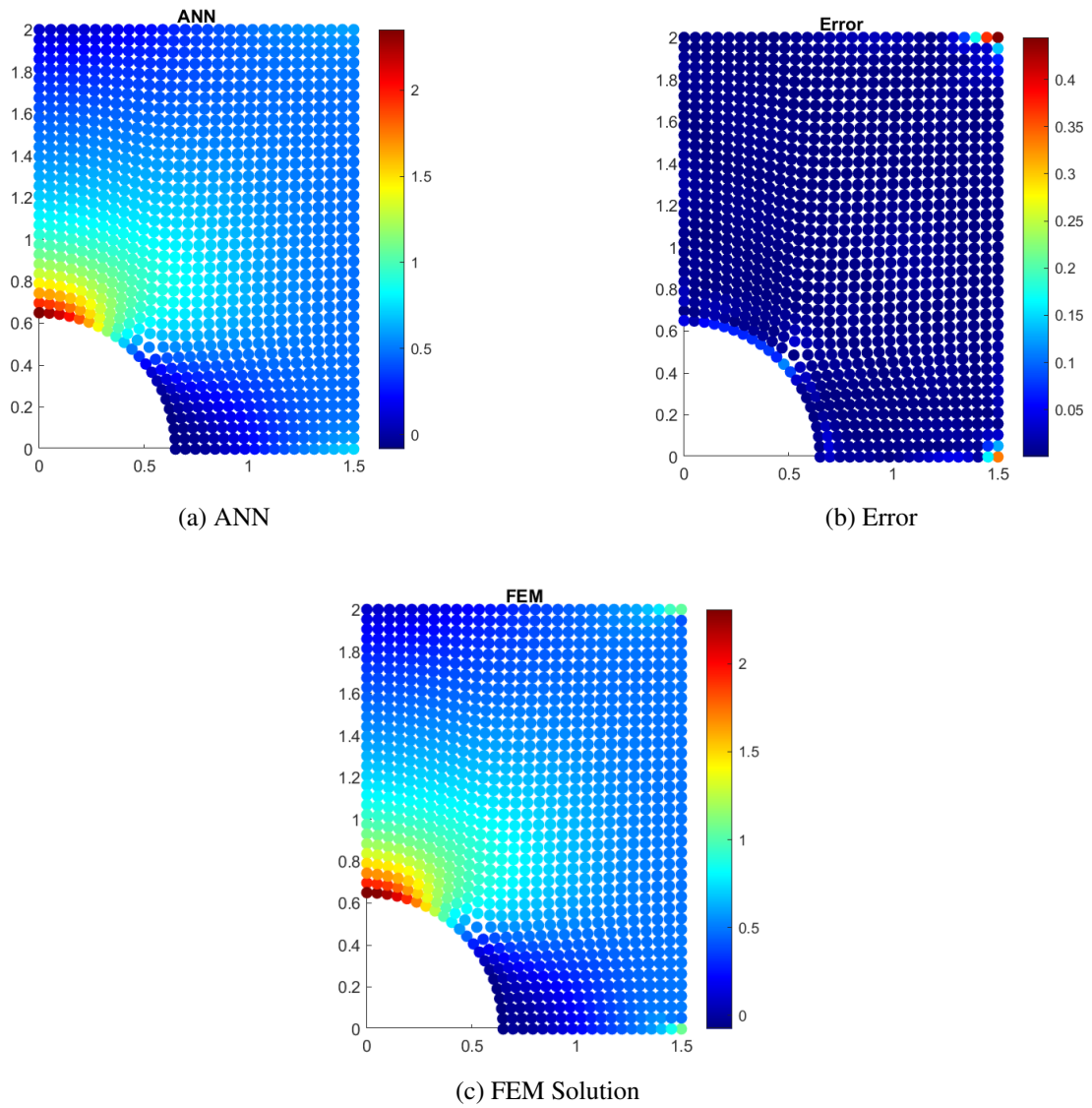


Figure 4.37: Predictions at the nodes for stress σ_{xx}

4.3.4.2 Stress σ_{yy}

For the stress σ_{yy} , the predictions for a certain plate dimension can be seen in Figure 4.38. The predicted stresses made by the ANN can effectively resemble the actual values.

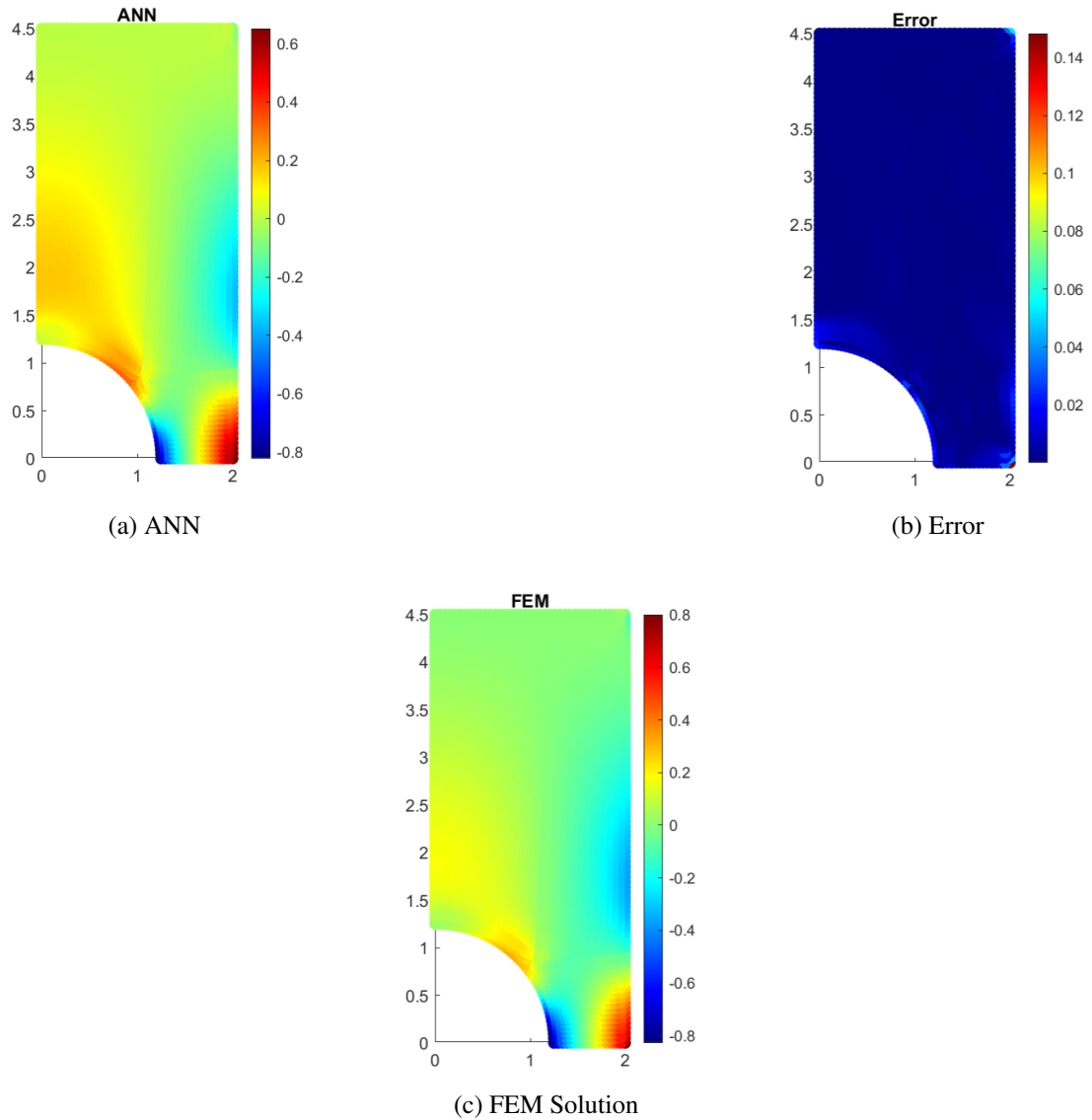


Figure 4.38: Predictions at the nodes for stress σ_{yy}

4.3.4.3 Stress τ_{xy}

For the shear stress τ_{xy} , the predictions are shown in Figure 4.39. Similar to the previous stress components, the neural network demonstrates a high level of accuracy in predicting τ_{xy} at the nodes.

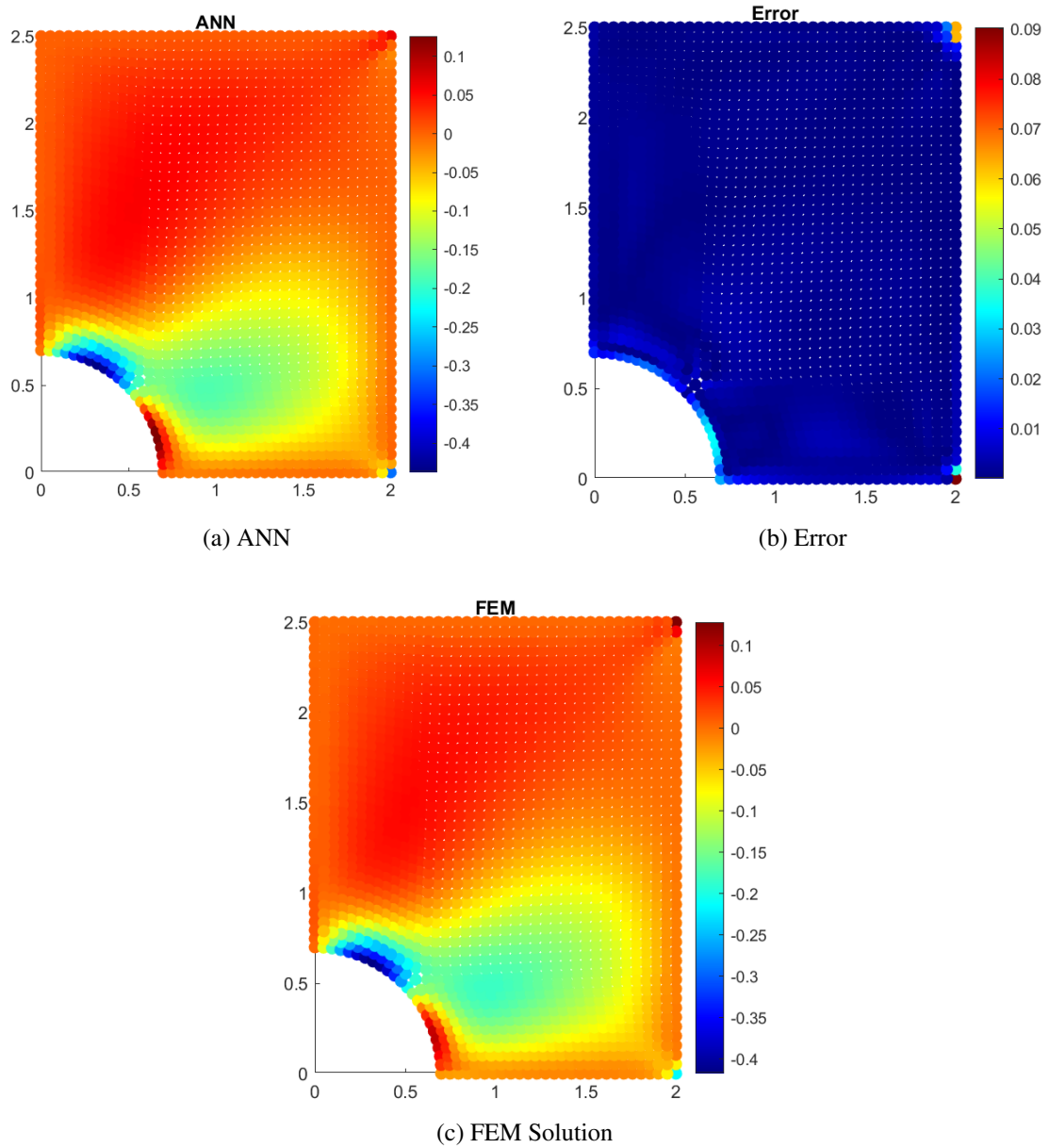


Figure 4.39: Predictions at the nodes for stress τ_{xy}

The error plot for the shear stress component closely resembles the error plots for the stress components at the x and y direction. The more pronounced errors are situated at the top right corner of the plate.

4.3.5 Strain at Nodes

The trained network was also used to predict the strain at the nodes, using the model originally trained on integration points. The results for each strain component are presented below.

4.3.5.1 Strain ϵ_{xx}

Figure 4.40 shows the predictions for the strain in the x direction, ϵ_{xx} . The analysis of the figure indicates that the neural network effectively predicts ϵ_{xx} with a good accuracy.

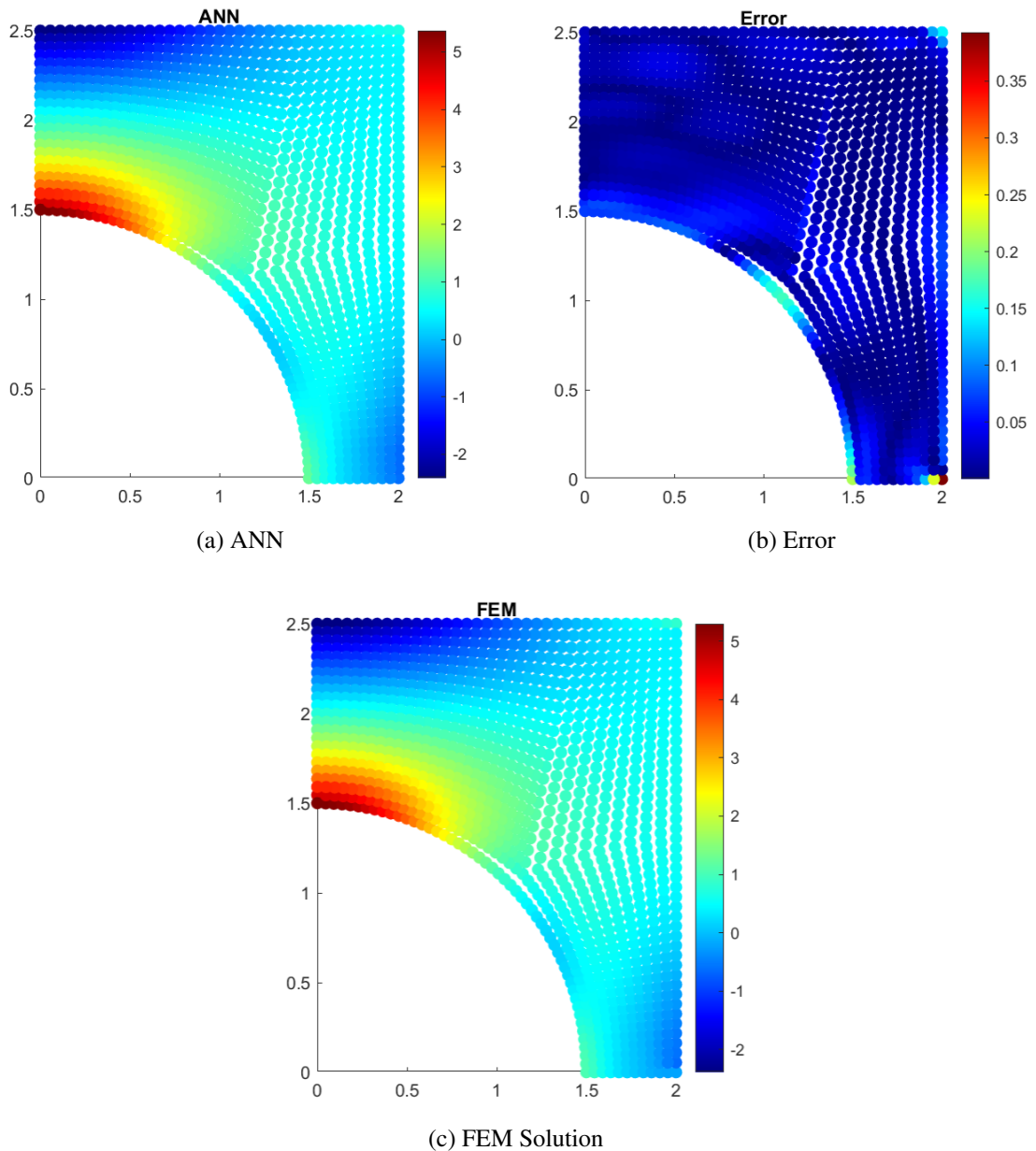


Figure 4.40: Predictions at the nodes for strain ϵ_{xx}

4.3.5.2 Strain ε_{yy}

For the strain in the y direction, ε_{yy} , the predictions are shown in Figure 4.41. The ANN closely matches the true values, resulting in very small errors.

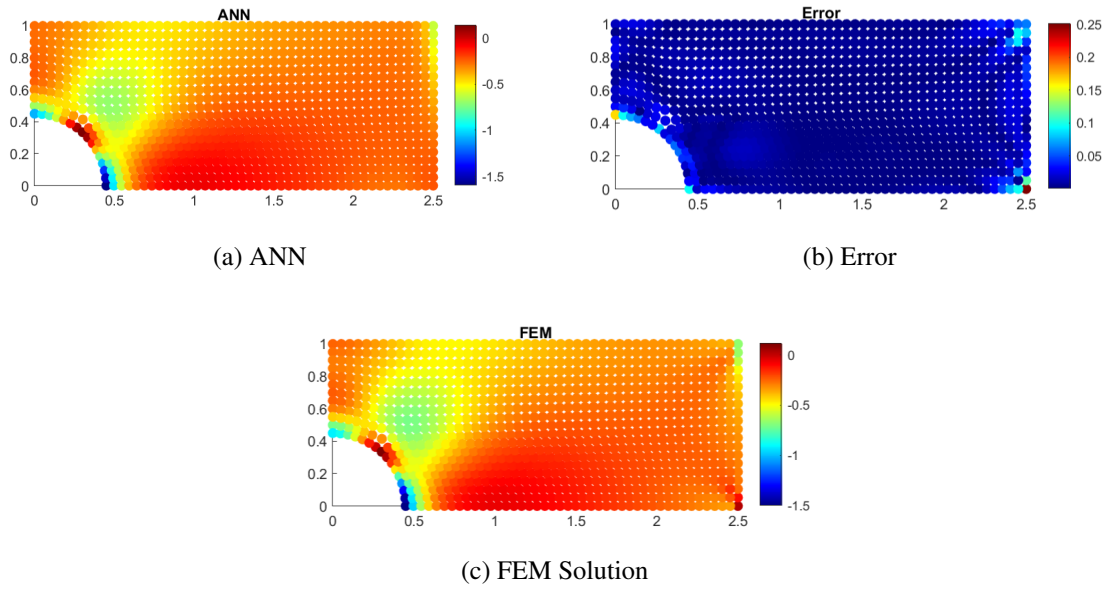


Figure 4.41: Predictions at the nodes for strain ε_{yy}

4.3.5.3 Strain ϵ_{xy}

Finally, the predictions for the shear strain, ϵ_{xy} , are presented in Figure 4.42. The ANN demonstrates reliable predictions for this strain component as well.

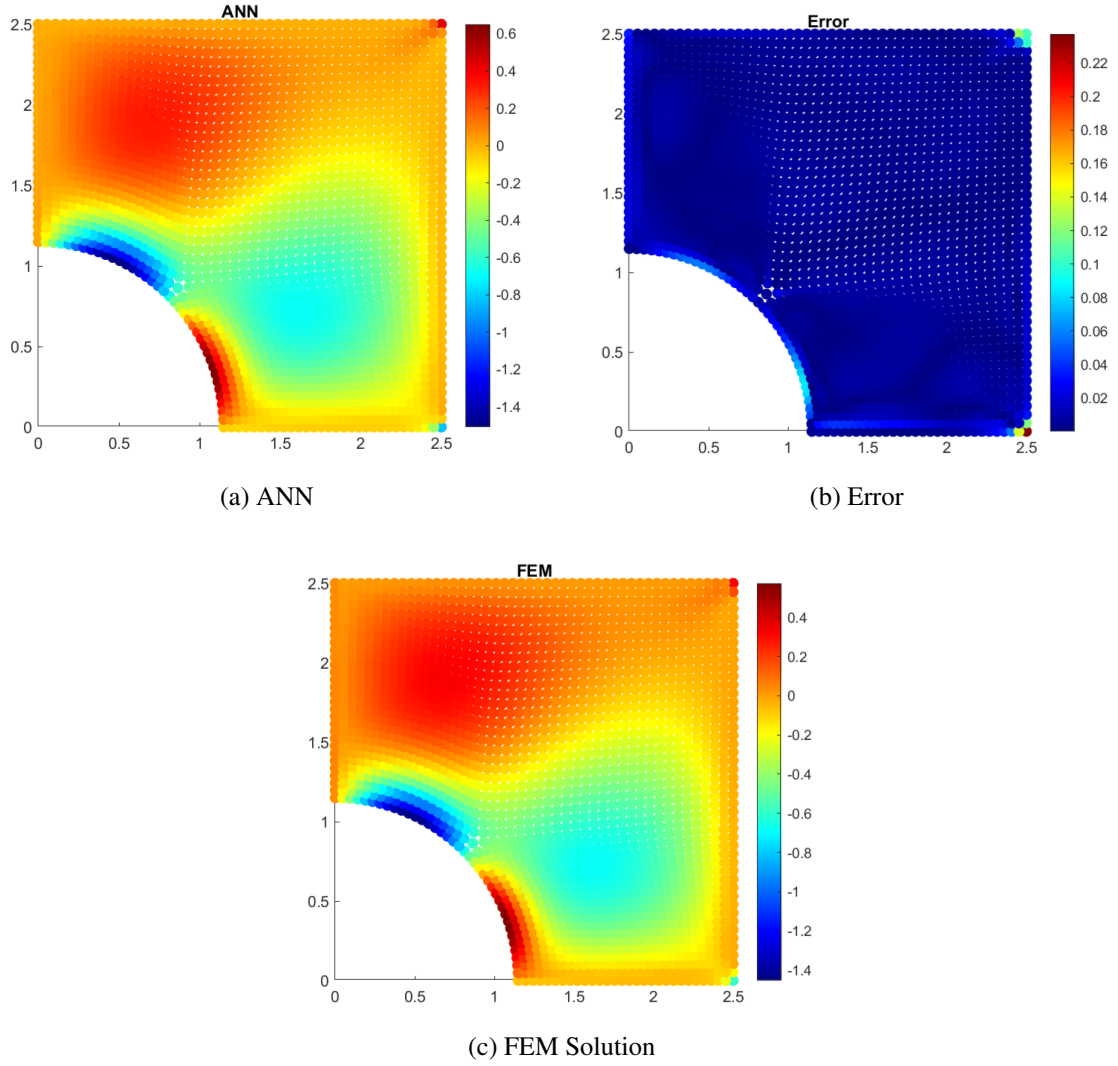


Figure 4.42: Predictions at the nodes for strain ϵ_{xy}

Similarly to what happened for the stress, the error plot for the strain in all directions shows very low values, indicating that the neural network trained on integration points can effectively predict strain at the nodes.

4.4 Discussion

Before delving into the detailed discussion of the results, it is important to highlight a key aspect of the hyperparameter sensitivity analysis: the results represent an average across several models. Consequently, for example while there may be individual cases where models using the *ReLU* activation function outperform those using *tanh*, the overall performance trends indicate that the *tanh* activation function consistently yields better results.

The results show that the *tanh* activation function is superior to *ReLU* for predicting nodal displacements. This trend is particularly evident in models trained without outliers, where the *tanh* function exhibits smoother training curves and more consistent performance. However, when outliers are present in the training data, the *ReLU* function seems to handle the perturbations slightly better, leading to a smaller but noticeable improvement in performance. Despite this, the overall error in models trained with outliers is generally higher, as illustrated by the predictions where the error distribution becomes more erratic when outliers are included.

The validation loss across epochs, particularly in Figure 4.10, back up this point. The *tanh* activation function not only promotes more stable learning but also mitigates sudden fluctuations during training, particularly in datasets without outliers. This characteristic makes it an excellent choice for predicting displacements, as it is capable of managing both positive and negative values and provides a stable gradient throughout the learning process.

For both of the displacement predictions, u and v , models trained without outliers and using the *tanh* activation function consistently yielded the best results.

Based on the results from nodal displacement predictions, it was decided to use the *tanh* activation function for the stress predictions at integration points. This decision was also driven by the need to save computational time, as *tanh* had already demonstrated superior performance in previous experiments.

The stress predictions, particularly for the normal stresses σ_{xx} and σ_{yy} , followed a similar pattern to the displacement predictions. The models performed significantly better when outliers were removed from the dataset, and a learning rate of 0.0001 yielded the most accurate results, as shown in Figures 4.16 and 4.20. Both models exhibited pronounced errors at the top and bottom right corners of the plate.

When comparing the models with and without outliers for normal stress predictions, the instability introduced by outliers was clearly visible. Models trained with outliers struggled to maintain stable learning curves, particularly in predicting σ_{xx} and σ_{yy} , as evidenced in Figures 4.15 and 4.19. However, when predicting the shear stress, τ_{xy} , the model with outliers performed similarly to the model without outliers, with the MAE across epochs showing nearly identical learning curves, as depicted in Figure 4.23. This indicates that the presence of outliers has a lesser impact on the model's ability to predict shear stress accurately.

For strain predictions, the presence of outliers had a notably smaller impact compared to displacements and stresses. The models without outliers consistently achieved slightly lower absolute

errors, but the overall performance gap between models with and without outliers was less pronounced. This suggests that strain predictions are less sensitive to the presence of anomalous data compared to other outputs, although removing outliers still contributes to marginally better performance.

In general, the presence of outliers negatively impacted the performance of the neural network across all outputs. The *tanh* activation function consistently outperformed *ReLU* in most cases, especially when outliers were excluded. The areas where the model struggled the most were around the hole where the stress concentration was higher and near the top right corner. These areas of high stress likely pose additional challenges for both the FEM and ANN due to the complexity of the mechanical response in these regions.

After identifying the optimal hyperparameters for predicting stress and strain components at the integration points, the next step was to apply these hyperparameters to predict stresses and strains at the nodes. The neural network, trained on integration point data, performed remarkably well in predicting nodal values for both stress and strain components.

The high level of accuracy achieved in nodal predictions indicates that the neural network was capable of effectively approximating the FEM's nodal values. This is particularly significant because it demonstrates that the trained network can generalize beyond its initial training domain at the integration points, providing reliable predictions at the nodes. The ability to extend predictions in this manner showcases the robustness of the model and its potential as a surrogate for FEM in broader applications.

Chapter 5

Plate with four identical holes

In this chapter, the analysis focuses on a plate with four holes, using an artificial neural network (ANN) to predict the structural response. The problem is first outlined, followed by a detailed description of the methodology employed. Similar to the previous case, the same outputs - displacement, stress, and strain - are examined. However, this time, the optimal set of hyperparameters previously identified for each output in the initial case is applied to assess their effectiveness. The accuracy of these hyperparameters is evaluated by making predictions on unseen data, providing insight into the model's generalisation capability. As before, the model trained on integration points will be tested to determine its effectiveness in predicting stresses and strains at the nodes.

5.1 Problem Description

In addition to the previous case of a plate with a hole in the center, a plate with four identical holes subjected to a distributed load was also analysed. The following Figure 5.1 is a clear representation of this particular case. The variables of this problem include the length L , width W , and radius r of the holes, as well as the x and y coordinates of the holes.

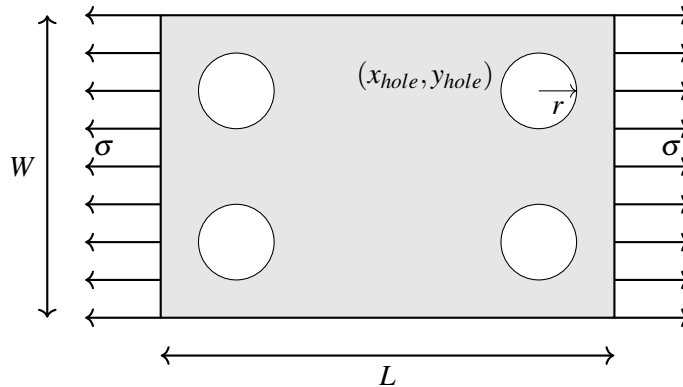


Figure 5.1: Plate with four holes

The distributed load applied to each side of the plate is also the same as the previous case. The load applied to each node is given by 1 N (*Newton*) divided by the number of nodes present on the width of the plate.

As it was previously done, this second case will be studied in terms of displacements, stresses and strains. In the first scenario involving a plate with a circular hole, a neural network was trained with data from FEM in order to predict the displacements, stresses, and strains. For each parameter under study, an optimal set of hyperparameters was identified, leading to the best model performance.

In this second case, the same outputs as before will be investigated, and the optimal hyperparameters previously determined for each output will be used to validate their accuracy in this new context. Given the significant impact of outliers on model performance and prediction accuracy, special attention was paid to the aspect ratio of the holes' radius to the plate's length and width. This consideration helps reducing the quantity of outliers, thereby making the data more homogeneous and improving the reliability of the predictions.

5.2 Methodology

5.2.1 FEM Analysis

The plate, which is subjected to a distributed load, is firstly analysed using the Finite Element Method with the purpose of creating a dataset for training the neural network. Due to the plate's symmetry along the x and y axes, only one quarter of the plate is analysed.

Multiple FEM simulations were conducted using *Python* scripting in *Abaqus*. A *.csv* file was generated to store the values of the varying parameters used in these simulations. This file includes all the values for the length L , width W , radius r and the hole coordinates, x_{hole} and y_{hole} .

The length, as well as the width, is initialized at 1 mm and incremented by 0.5 mm until it reaches 5 mm. For each combination of length and width, the coordinate x of the hole (x_{hole}) increments from 0.0 mm to the current length in steps of 0.5 mm and, for each (x_{hole}), the coordinate y of the hole (y_{hole}) increments from 0.0 mm to the current width in steps of 0.5 mm. For each combination of x_{hole} and y_{hole} , the radius r of the hole increments from 0.2 mm to half of the smallest dimension in steps of 0.2 mm. Additionally, the hole must fit within specific boundaries of the plate, ensuring the distance to the plate edges is less than 5% of the plate's width or length. This process results in 4888 different parameter combinations, corresponding to the number of FEM simulations performed. The code pertaining to the *.csv* file with all parameter combinations is included in Annex A.7.

The two dimensional plate models, with their associated parameters and boundary conditions, were generated using the *Abaqus* software. All of the pre processing steps remain the same as the previous case:

- Young's Modulus (E) equal to 1 MPa and Poisson's ratio $\nu = 0.3$
- Meshing size of 0.05 mm

- Four node plane stress element (CPS4) - Element Type
- Symmetric boundary conditions
- Uniform distributed load

The meshed plate geometry as well as its boundary conditions can be seen in the following Figure 5.2. Additionally, the used scrip to perform the simulations is given in Annex A.8.

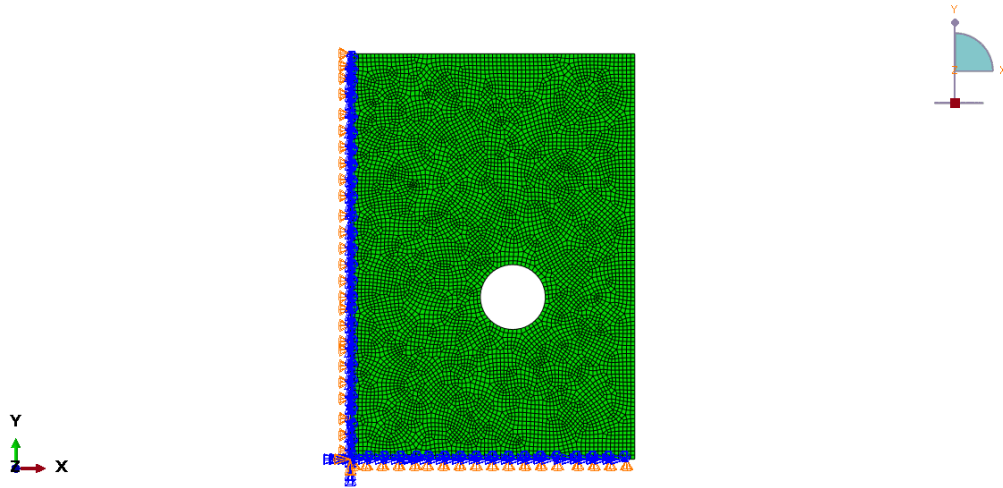


Figure 5.2: Plate Boundary Conditions

5.2.2 ANN Training

The data from the simulations is used to develop an Artificial Neural Network (ANN). The same neural network architecture used in the first case is employed here, with a key difference: the optimal hyperparameters for predicting displacements, stresses, and strains in the plate with four holes have already been determined based on the first case study.

Given the increased complexity and volume of data for the plate with four holes, the number of epochs for training the neural network was increased from 1000 to 2000. This adjustment was made after observing and verifying the validation loss trends during training. The additional epochs allow the model to learn more effectively from the data, leading to significantly improved performance and more accurate predictions. The code relative to the train of the neural network is provided in Annex A.9.

5.3 Results

This section presents the results achieved by the neural network in predicting the node displacements and, stresses and strains at both the integration points and nodes. The optimal set of hyperparameters identified in the first case will be applied and tested in this new scenario to validate their accuracy and effectiveness. By using these previously determined hyperparameters, it is aimed to assess the consistency and robustness of the model's predictive capabilities across different cases.

5.3.1 Node Displacements

The node displacements predictions for both the x and y directions (u and v , respectively) are the first outputs to be analysed in the neural network.

5.3.1.1 Node Displacement - x direction

The optimal results for predicting the displacement in the x direction in the first case were achieved using a specific set of hyperparameters. These same hyperparameters were applied to this new case, as follows:

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The hyperparameters mentioned above enable the network to achieve the following validation MAE over the epochs, as illustrated in Figure 5.3.

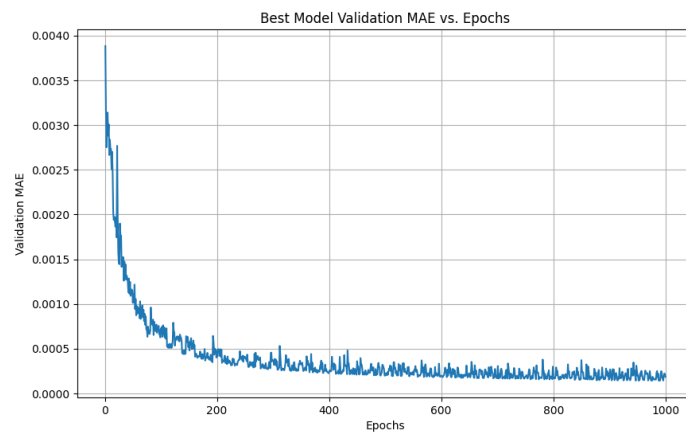


Figure 5.3: Validation MAE along 1000 epochs - u displacement

The Figure 5.3 shows the validation Mean Absolute Error (MAE) over 1000 epochs. In the first 200 epochs there is a significant reduction in the validation MAE, indicating rapid initial learning. From the 200 epochs to the 400 epochs it also decreases but at a slower rate and after it, it starts stabilising and fluctuating around a low value.

Overall, the Figure 5.3 highlights that the model is well trained, with an effective reduction in error over time and stability in the later stages of training. With this, it is possible to conclude that the chosen hyperparameters clearly provide an optimal setup for this specific case.

The predictions for the u displacement on the testing data using the neural network model are illustrated in Figure 5.4. These predictions were generated for various plate and hole dimensions.

The results indicate that the neural network, using the specified hyperparameters, has successfully learned to approximate the displacement field with high accuracy.

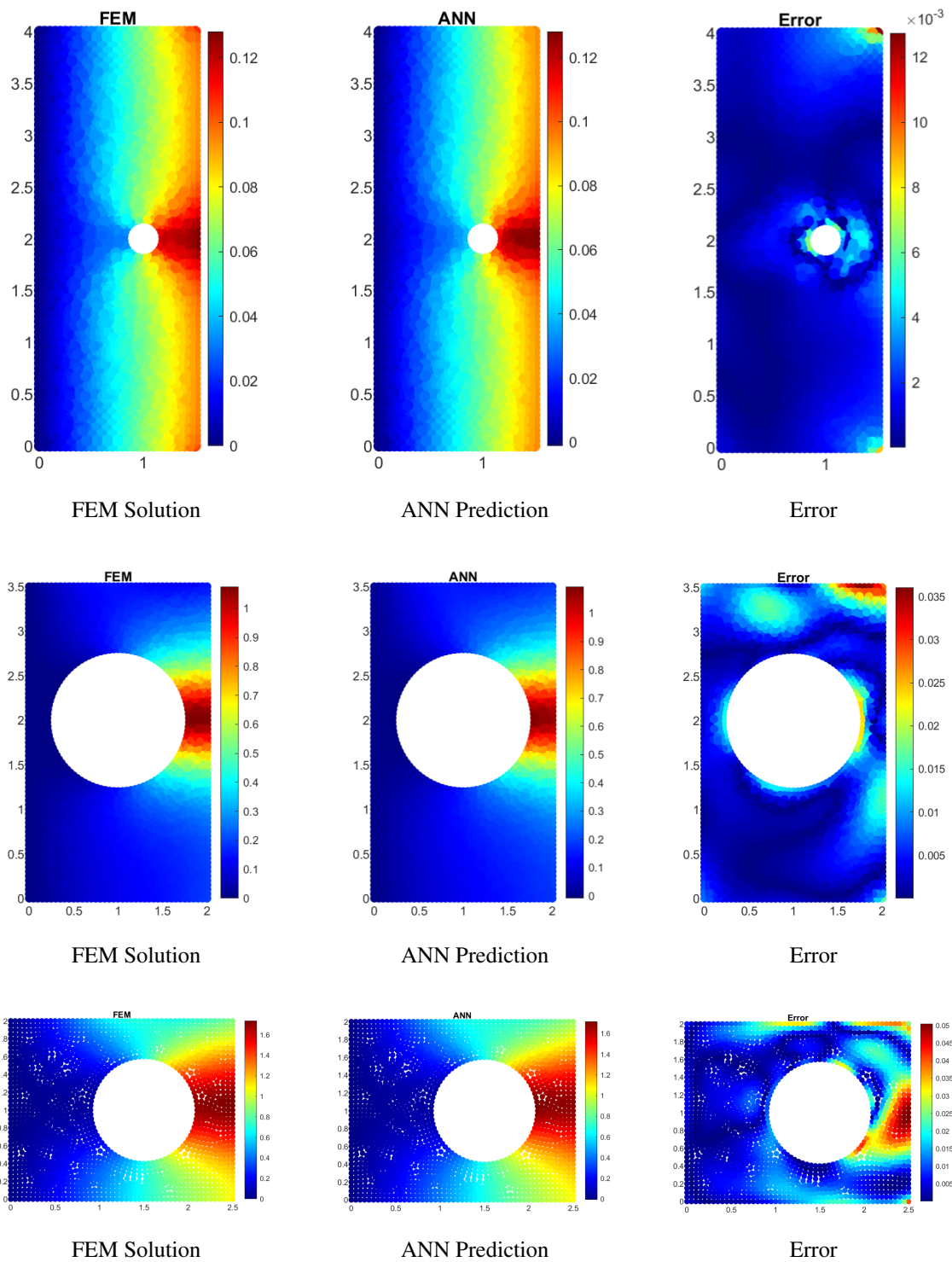


Figure 5.4: Comparison of FEM and ANN predictions for displacement u across three different plate dimensions

5.3.1.2 Node Displacement - y direction

The optimal set of hyperparameters from the first case was employed to predict the displacement v in the y direction. The hyperparameters are,

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The hyperparameters mentioned above enable the network to achieve the following validation MAE over the epochs, as shown in Figure 5.5.

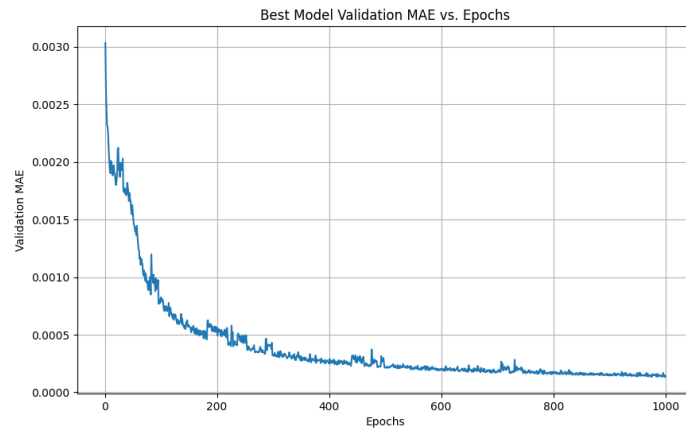


Figure 5.5: Validation MAE along 1000 epochs - v displacement

The Figure 5.5 illustrates the Mean Absolute Error (MAE) of the model predicting the v displacement over 1000 epochs. In the first 200 epochs the MAE decreases sharply indicating that the model is quickly learning. After this rapid decrease, the validation MAE continues to decline more gradually until it reaches a point where the error is minimum.

One potential improvement for analysing the subsequent parameters is to increase the number of epochs to 2000. As shown in Figure 5.5, the model demonstrates a trend towards further reduction in validation MAE, suggesting that additional training could enhance its performance. Moreover, given that this case involves more data compared to the first case, extending the training duration could allow the model to learn more effectively from the larger dataset, potentially leading to even better predictions.

The predictions of the model for the v displacement on the testing data are illustrated in Figure 5.6. The predictions were generated for different plate and hole dimensions.

It is worth noting that the displacements in the y direction are all negative, as demonstrated through the visualization of the figures. From the analysis, the ANN predictions closely resemble the FEM solutions demonstrating that the neural network effectively learns the prediction of the v displacement.

Additionally, the error plot shows very small discrepancies between the FEM solution and the ANN prediction suggesting that the ANN provides a highly accurate approximation.

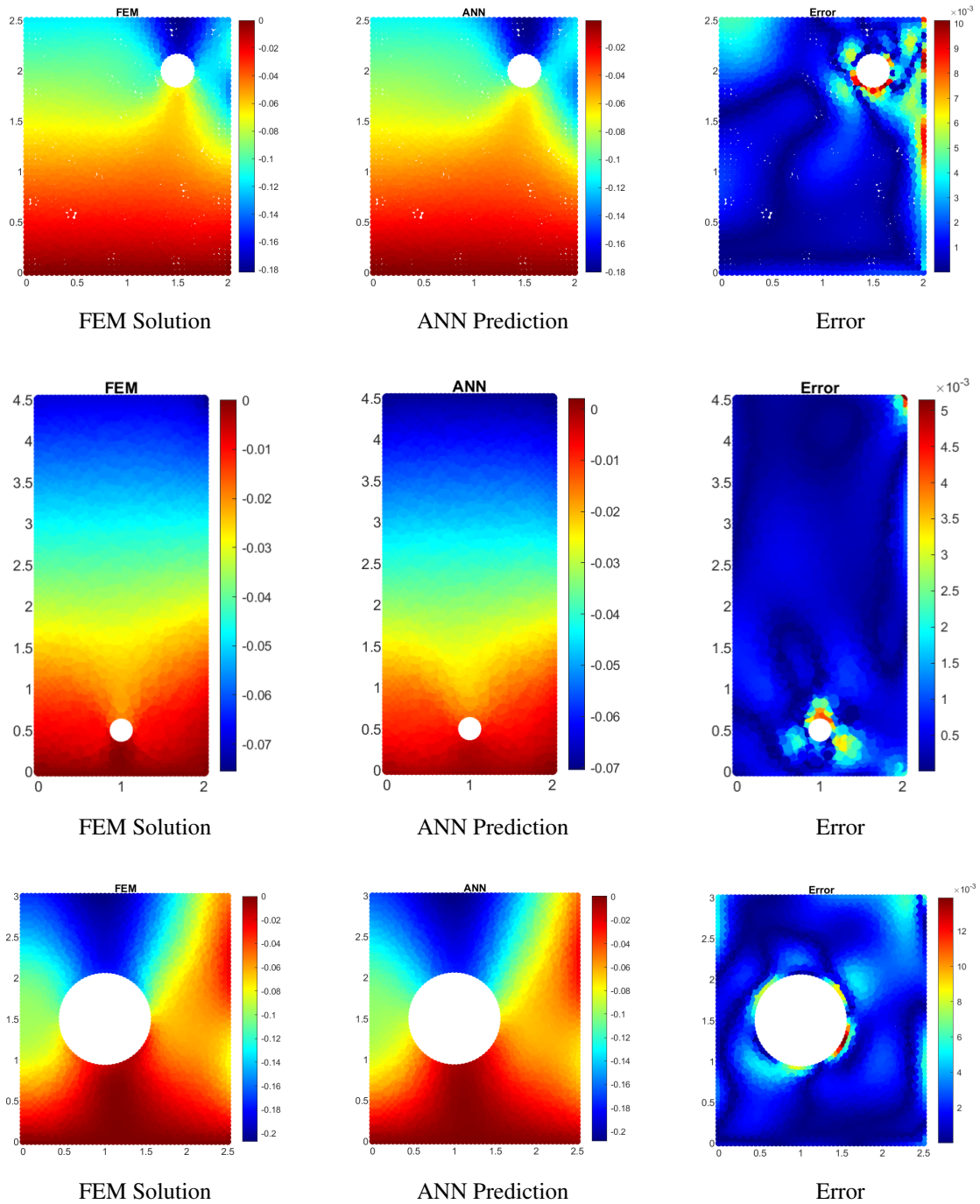


Figure 5.6: Comparison of FEM and ANN predictions for displacement v across three different plate dimensions

5.3.2 Stress at Integration Points

The next outputs to be studied for the plate with four holes were the stresses at the integration points. The stresses encompasses the stress at the x and y direction (σ_{xx} , σ_{yy}) and the shear stress (τ_{xy}).

5.3.2.1 Integration Point - Stress σ_{xx}

The first stress to be studied is the stress σ_{xx} . The same hyperparameters as the first case are used as follows,

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The validation MAE over the epochs for the previous set of hyperparameters can be seen in Figure 5.7.

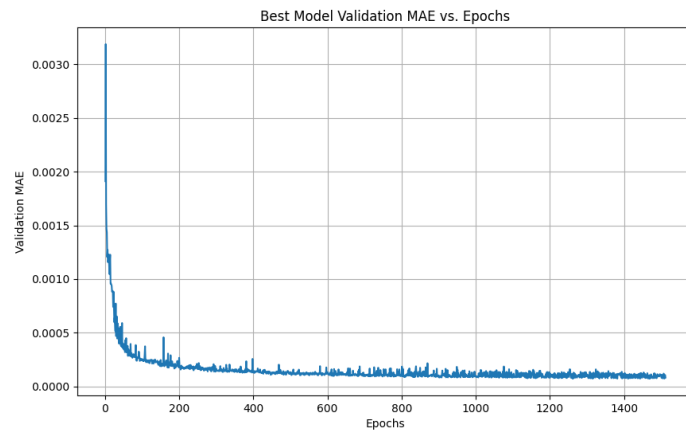


Figure 5.7: Validation MAE along 2000 epochs - Stress σ_{xx}

The Figure 5.7 shows the validation MAE across 2000 epochs. The model stops near the epoch 1500 due to the *EarlyStopping()* function that stops the training when the model is not improving anymore.

In the first 200 epochs, the model has a steep descent of the validation MAE and then continues to decrease but in a more gradual way until the error starts converging and stabilising around a minimal value. The overall performance of the model indicates that it is learning effectively.

The predictions for the stress in the x direction, σ_{xx} , on the testing data can be seen in Figure 5.8. The predictions were made for various plate and hole dimensions.

The results show that the specified hyperparameters can successfully predict the stress in the x direction. The error is generally low for the different predictions, suggesting that the ANN model has a high accuracy.

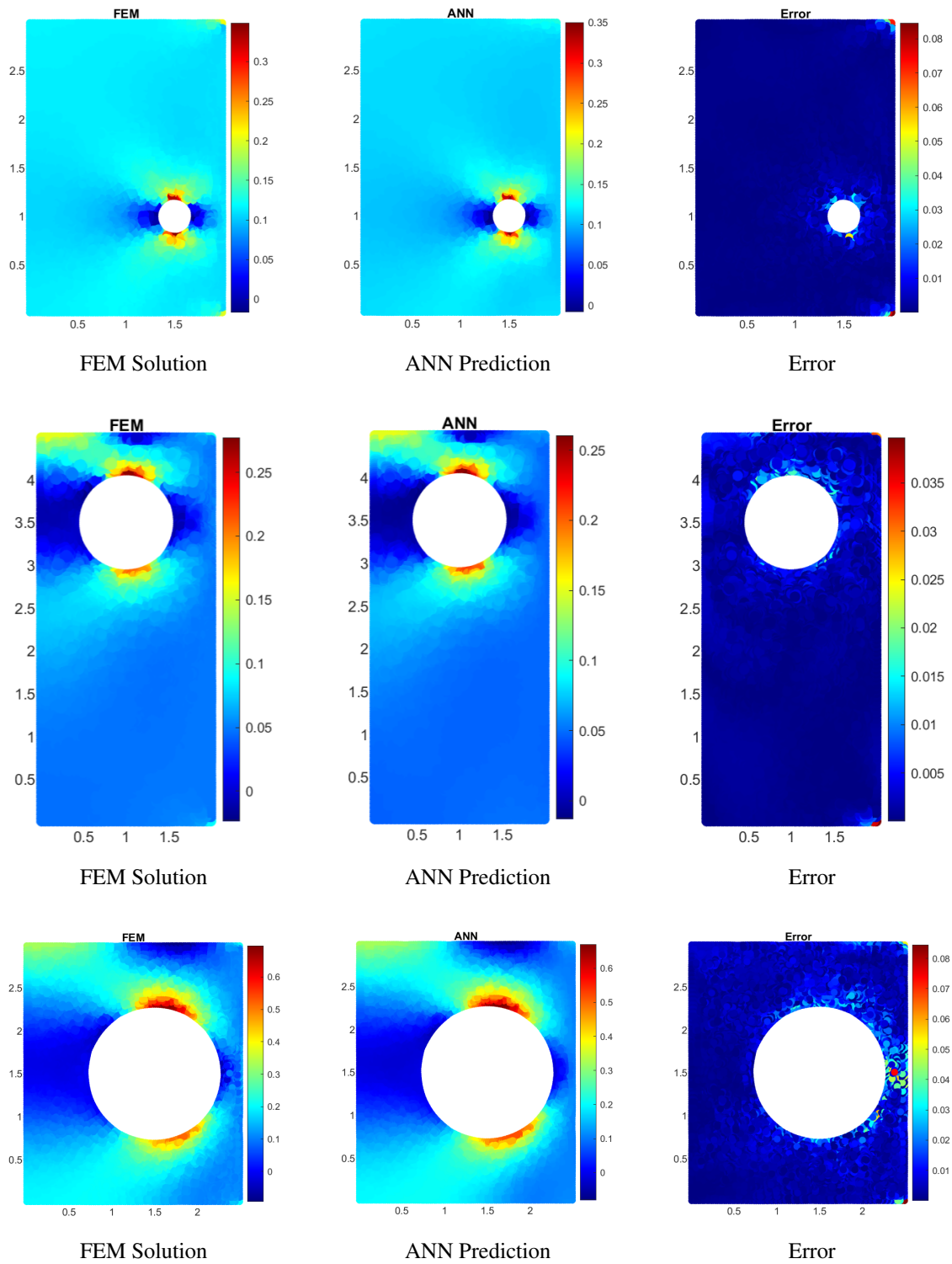


Figure 5.8: Comparison of FEM and ANN predictions for Stress σ_{xx} across three different plate dimensions

5.3.2.2 Integration Point - Stress σ_{yy}

The stress at the integration point in the y direction, σ_{yy} , is studied and the used hyperparameters to apply to the network are the following,

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 65536
- **Learning Rate:** 0.0001

The model's performance, reflected by the validation MAE over the epochs using the previously defined hyperparameters, is illustrated in Figure 5.9.

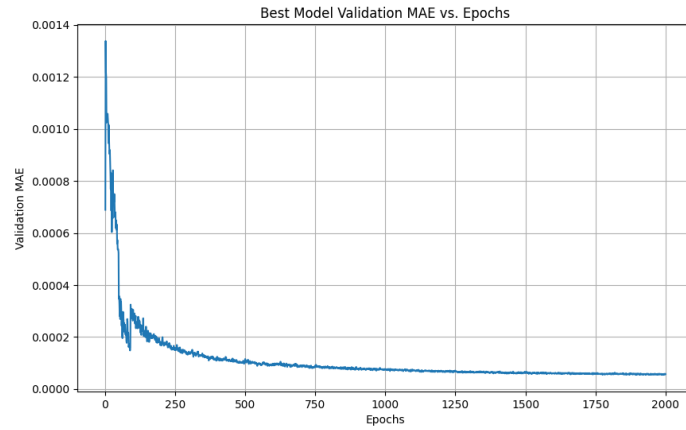


Figure 5.9: Validation MAE along 2000 epochs - Stress σ_{yy}

From the analysis of the Figure 5.9, it can be extracted that the model learns effectively showing a sharply decrease in the validation MAE over the first 500 epochs and then converging and stabilising at a low value.

The predictions for the stress in the y direction, σ_{yy} , made on the testing data are represented in Figure 5.10 for different plate geometries and radius.

From the Figure 5.10, it is possible to extract that the neural network model accurately predicts the stress in the y direction, σ_{yy} . The previously defined hyperparameters can predict the stress with minor errors highlighting the high accuracy of the model. The most prominent errors are the ones near the boundaries of the holes.

Concluding, the chosen hyperparameters are also as effective as they were for the first case, showing high accuracy for the predictions and generalising well.

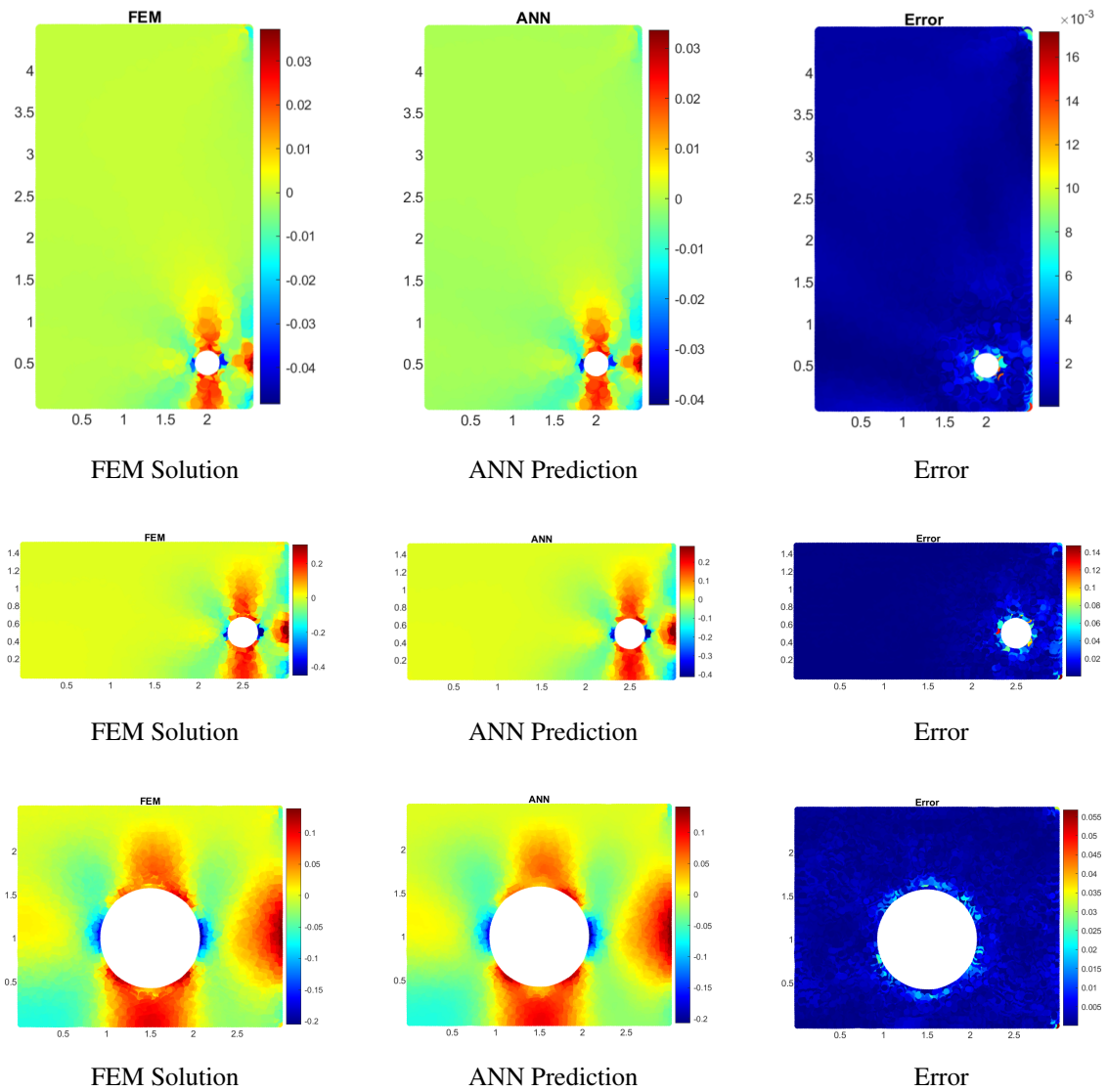


Figure 5.10: Comparison of FEM and ANN predictions for Stress σ_{yy} across three different plate dimensions

5.3.2.3 Integration Point - Stress τ_{xy}

The shear stress component in the xy plane, τ_{xy} , was studied. The predictions were made using the same hyperparameters as the first case and the same are given by,

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The validation MAE of the model over 2000 epochs with the above hyperparameters is represented in Figure 5.11.

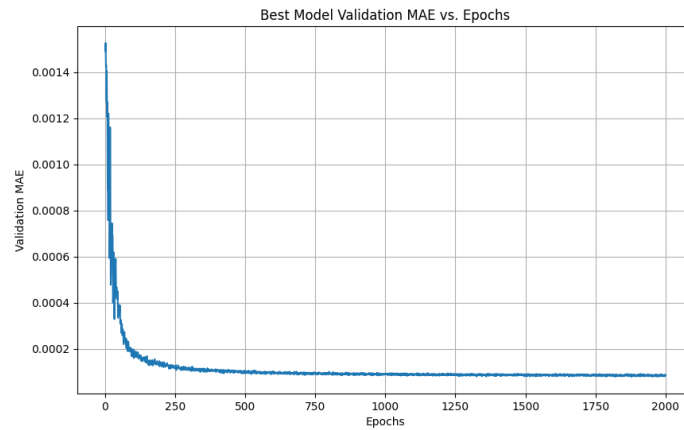


Figure 5.11: Validation MAE along 2000 epochs - Stress τ_{xy}

From the Figure 5.11, showing the performance of the model, it is evident that the model exhibits an effective and stable learning. The validation loss decreases rapidly over the first 250 epochs before stabilising at a very low value.

The predictions for the shear stress τ_{xy} were made and the results presented in Figure 5.12.

The ANN predictions closely resemble the FEM solutions in all the demonstrated cases, indicating that the neural network has effectively learned. The error map shows that the discrepancies between the ANN predictions and the FEM solutions are generally low and, the higher errors can be seen near the plate holes and in regions with sharp stress gradients.

In summary, the Artificial Neural Network with the chosen hyperparameters demonstrates strong predictive capabilities for the shear stress τ_{xy} across different plate geometries. The choice of the hyperparameters has proven to be effective.

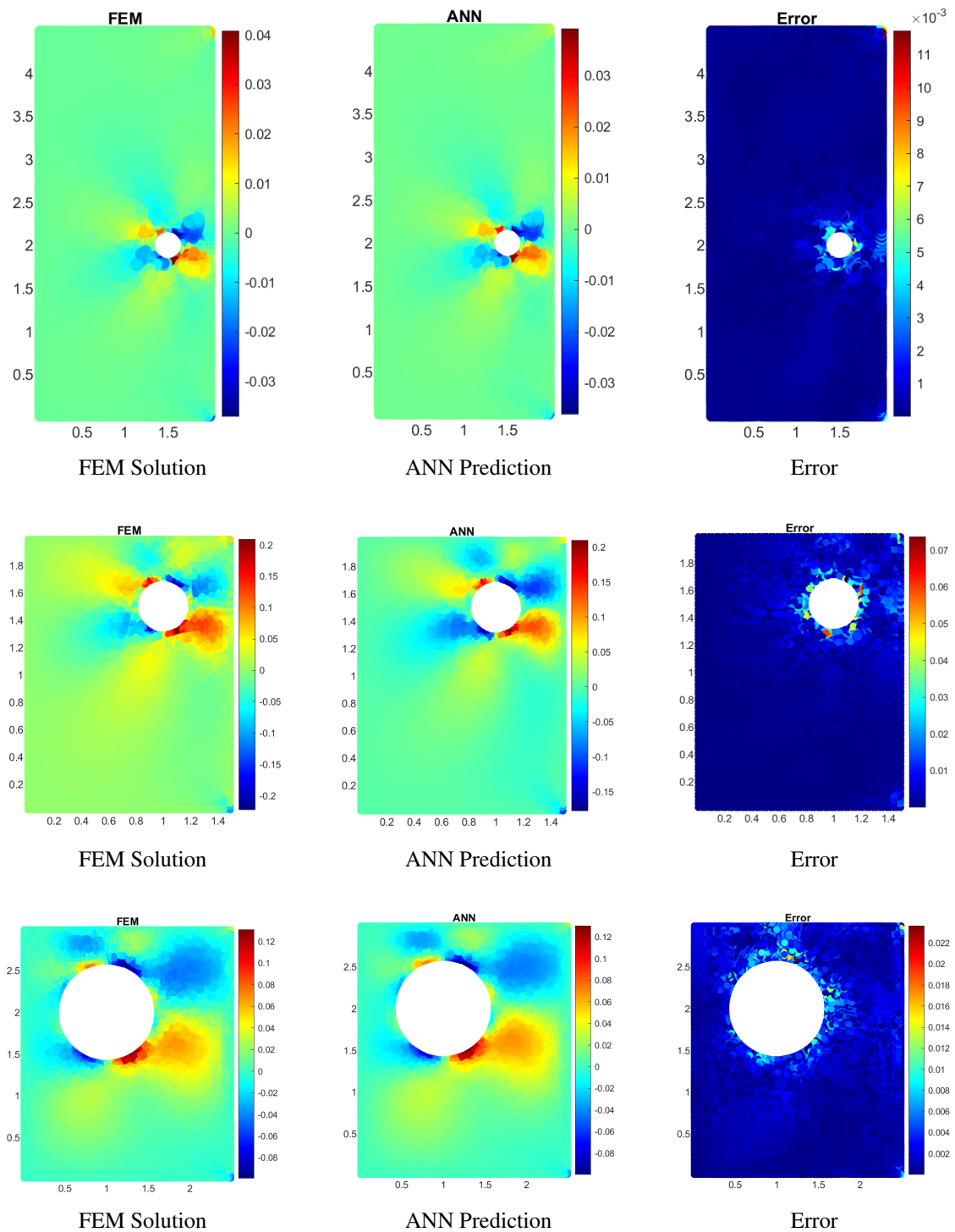


Figure 5.12: Comparison of FEM and ANN predictions for Shear Stress τ_{xy} across three different plate dimensions

5.3.3 Strain at Integration Points

The following outputs to be studied are the strains at the integration points. These strains include the strain at the x and y direction, ϵ_{xx} and ϵ_{yy} , respectively and the shear strain ϵ_{xy} .

5.3.3.1 Integration Point - Strain ϵ_{xx}

The first component of strain to be studied at the integration point is the ϵ_{xx} . The best set of hyperparameters that was used to predict this same strain for the first case was also used for this case. They are as follows,

- **Activation Function:** $\tanh$
- **Number of Hidden Layers:** 30
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The performance of the model, given by the validation MAE across the epochs, with the previous hyperparameters is represented in Figure 5.13.

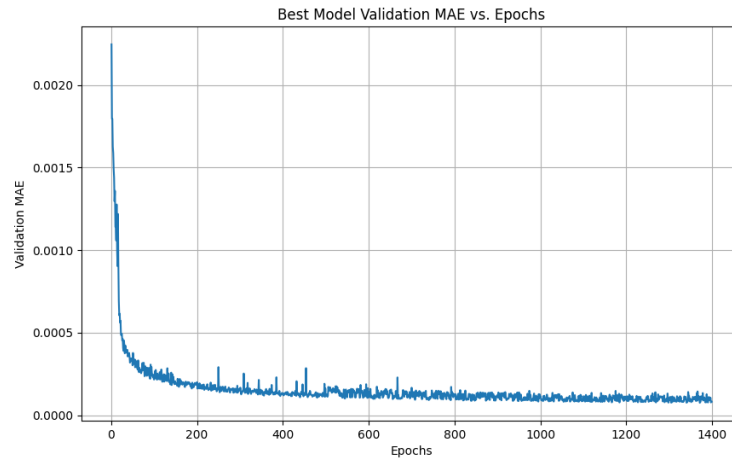


Figure 5.13: Validation MAE across the epochs - Strain ϵ_{xx}

The Figure 5.13 illustrates the validation loss of the model over the course of training epochs for predicting the strain in the x direction, ϵ_{xx} . Initially, the validation loss declines sharply within the first 200 epochs, after which it gradually stabilises and converges to a minimum value. The training process terminates around 1400 epochs, triggered by the *EarlyStopping()* function. This behavior suggests that the model has undergone stable training.

The predictions generated on the unseen test data are presented in Figure 5.14. These predictions, made using *Matlab*, demonstrate the model's performance across various plate geometries

and hole radius, highlighting its ability to generalize effectively to different structural configurations.

From the Figure 5.14 comparing the Finite Element Method solution and the Artificial Neural Network predictions for the strain ϵ_{xx} some observations can be made. The ANN predictions closely match the FEM solutions, indicating that the neural network has effectively learned to approximate the strain distribution in the plates.

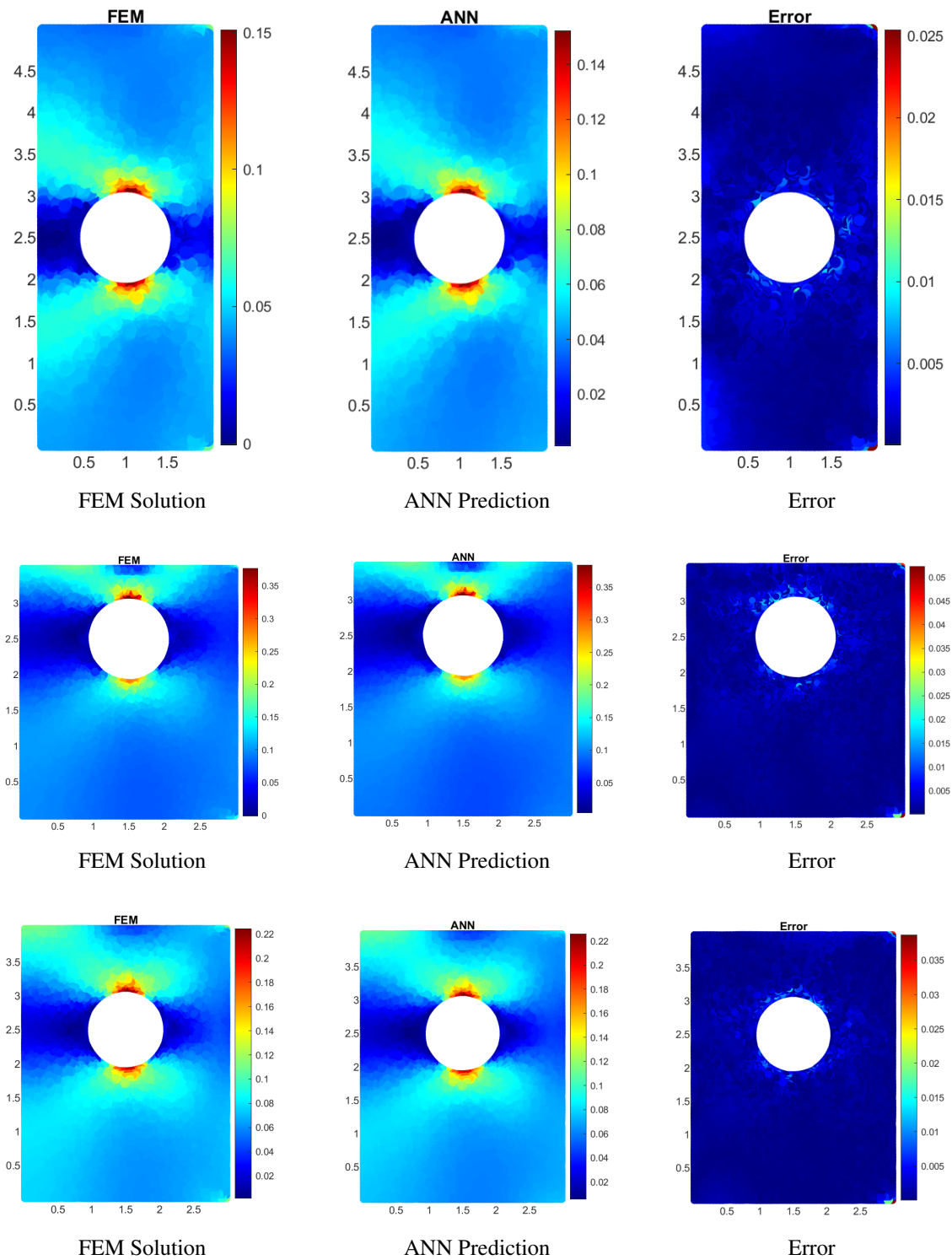


Figure 5.14: Comparison of FEM and ANN predictions for Strain ϵ_{xx} across three different plate dimensions

5.3.3.2 Integration Point - Strain ϵ_{yy}

The next component of strain to be studied is the strain in the y direction, ϵ_{yy} . The optimal set of hyperparameters used to predict the same strain for the first case was also used for this case and are as follows,

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 10
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The validation MAE across the epochs for the previously enumerated hyperparameters is given in Figure 5.15.

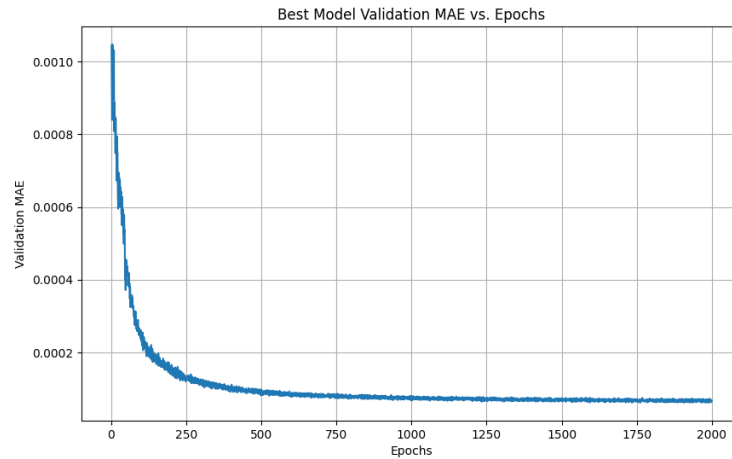


Figure 5.15: Validation MAE across the epochs - Strain ϵ_{yy}

The Figure 5.15 indicates that the neural network exhibits stable learning, with the validation MAE rapidly decreasing in the initial epochs and then gradually converging to a minimal value. This suggests that the network effectively captures the underlying patterns in the data.

Figure 5.16 presents the model's predictions on the testing data, revealing a strong alignment with the FEM solutions. Although the error is minimal across most regions of the plate, there is a slight increase in error near the hole.

Overall, the selected hyperparameters demonstrate robust performance and generalization capabilities. This suggests that the network configuration could be effectively applied to similar tasks and problems.

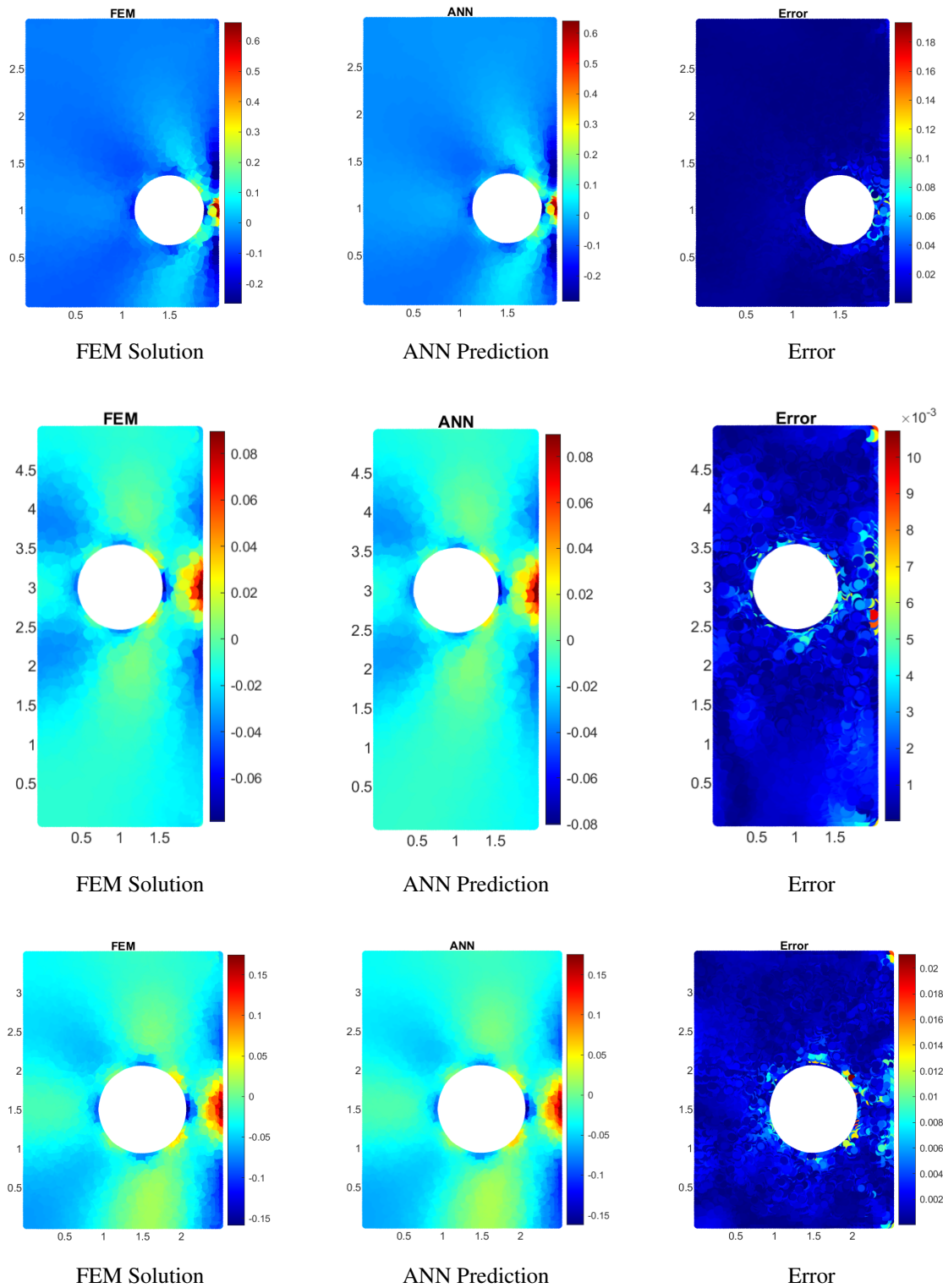


Figure 5.16: Comparison of FEM and ANN predictions for Strain ϵ_{yy} across three different plate dimensions

5.3.3.3 Integration Point - Strain ϵ_{xy}

The shear strain ϵ_{xy} at the integration point was also analysed. The best set of hyperparameters is the following,

- **Activation Function:** *tanh*
- **Number of Hidden Layers:** 20
- **Hidden Layer Size:** 32 neurons per layer
- **Batch Size:** 32768
- **Learning Rate:** 0.0001

The performance of the model, featuring the validation MAE across the epochs is given in Figure 5.17.

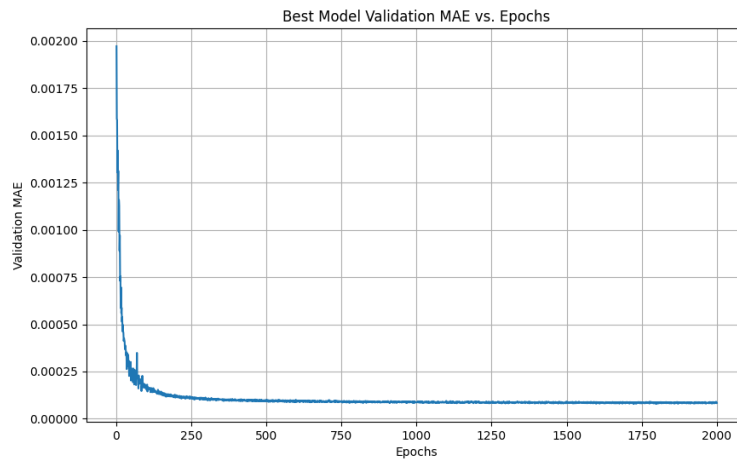


Figure 5.17: Validation MAE across the epochs - Strain ϵ_{xy}

The Figure 5.17 illustrates a rapid decline in MAE during the initial epochs, demonstrating that the neural network efficiently learns to minimise error and enhance prediction accuracy. After about 250 epochs, the MAE plateaus and gradually decreases, eventually converging to a low value, indicating the model's stability and effective learning process.

The comparison between the model's predictions and the FEM solution, shown in Figure 5.18, reveals that the neural network is highly effective in predicting the shear strain ϵ_{xy} .

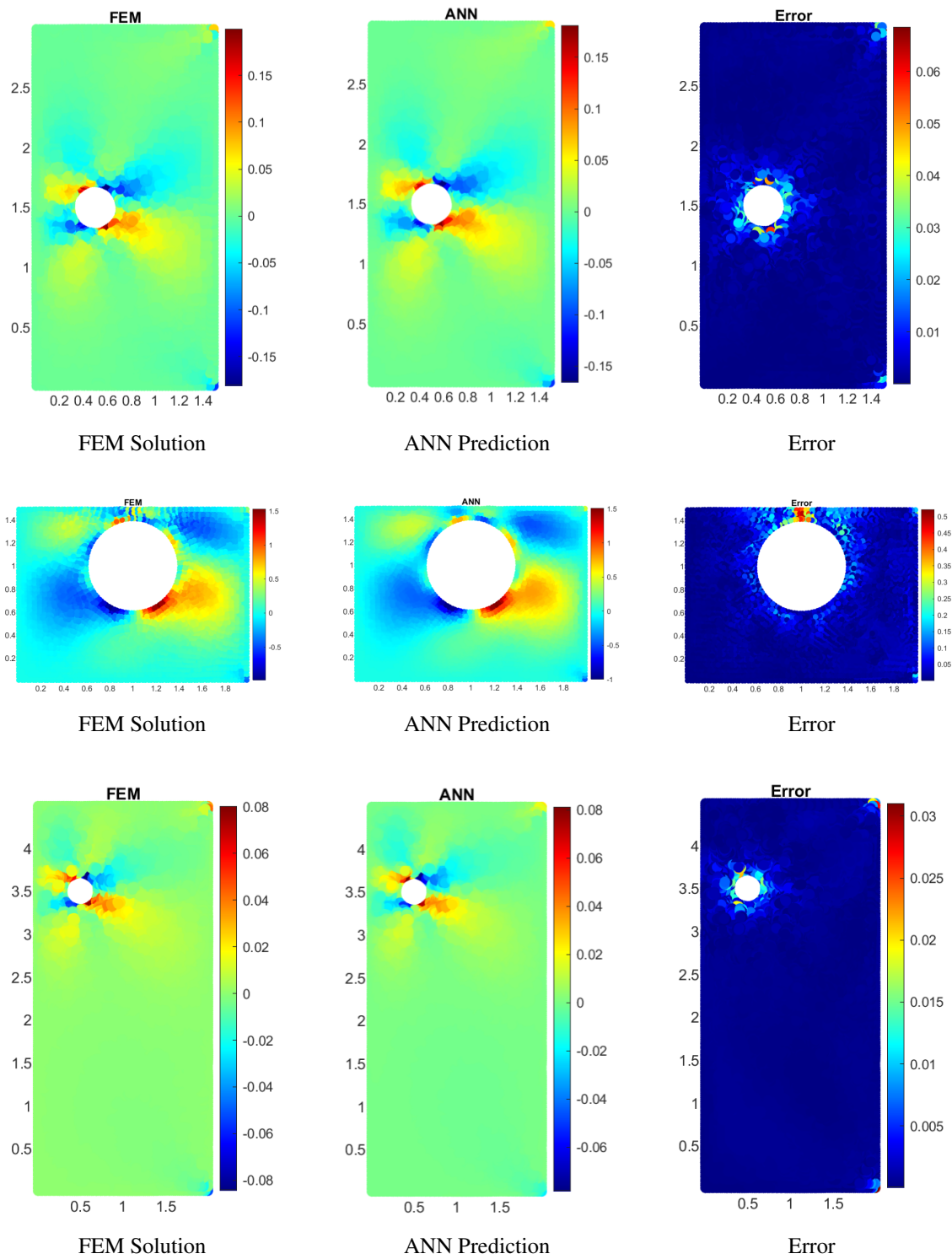


Figure 5.18: Comparison of FEM and ANN predictions for Strain ϵ_{xy} across three different plate dimensions

5.3.4 Stress at Nodes

The neural network, which was originally trained on the integration points, was also applied to predict the stresses at the nodes. The primary objective of this approach was to evaluate the model's accuracy in extrapolating its predictions from the integration points to the nodes. This test also serves as a critical step in determining the robustness and generalization capabilities of the neural network across different but related datasets.

5.3.4.1 Stress σ_{xx}

Figure 5.19 shows the predictions at the nodes for the stress in the x direction, σ_{xx} . From the analysis of the figures, the neural network effectively predicts the stress σ_{xx} .

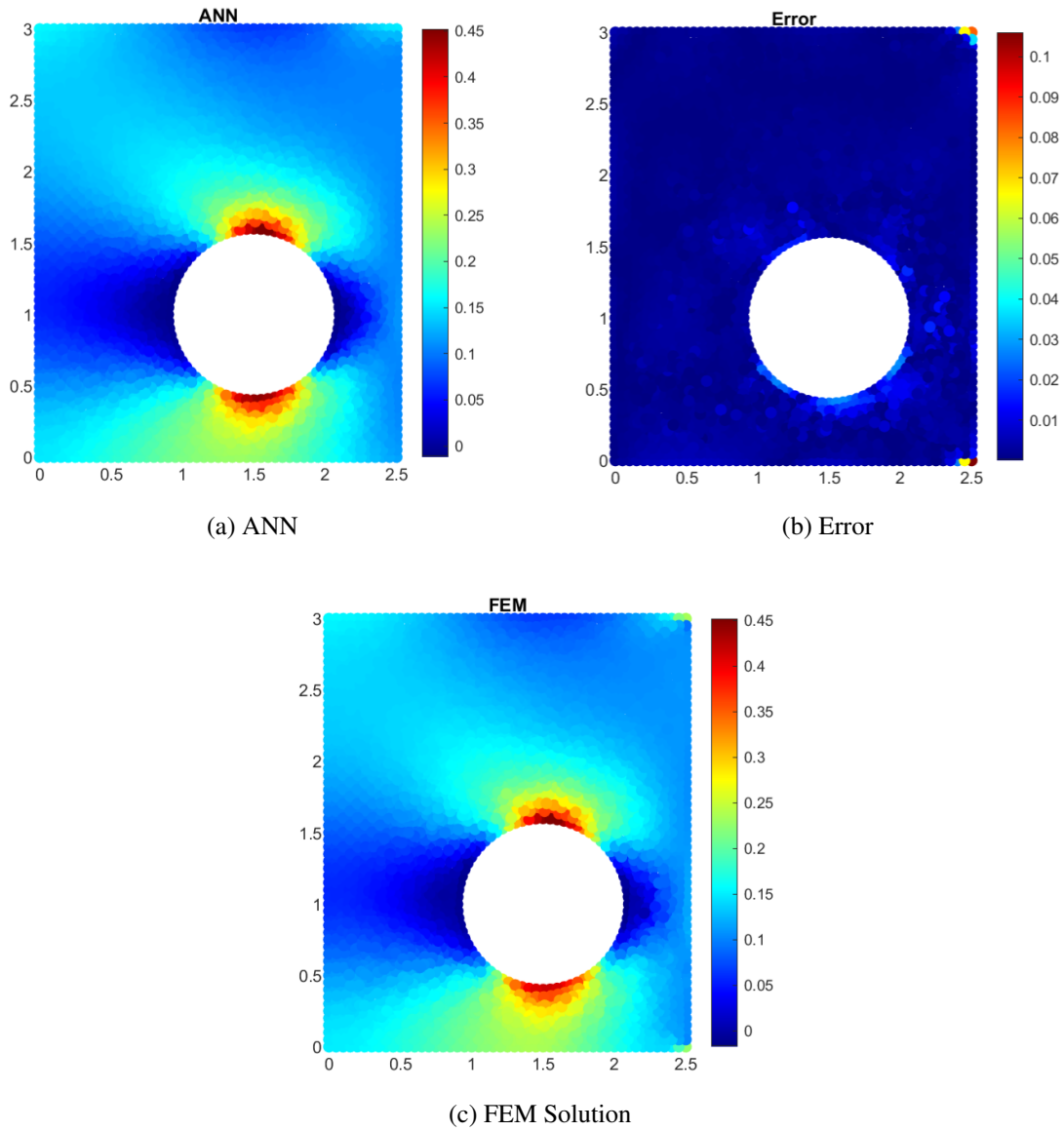


Figure 5.19: Predictions at the nodes for stress σ_{xx}

5.3.4.2 Stress σ_{yy}

The Figure 5.20 illustrates the predictions for the stress σ_{yy} . The values predicted by the neural network closely resemble the FEM solution.

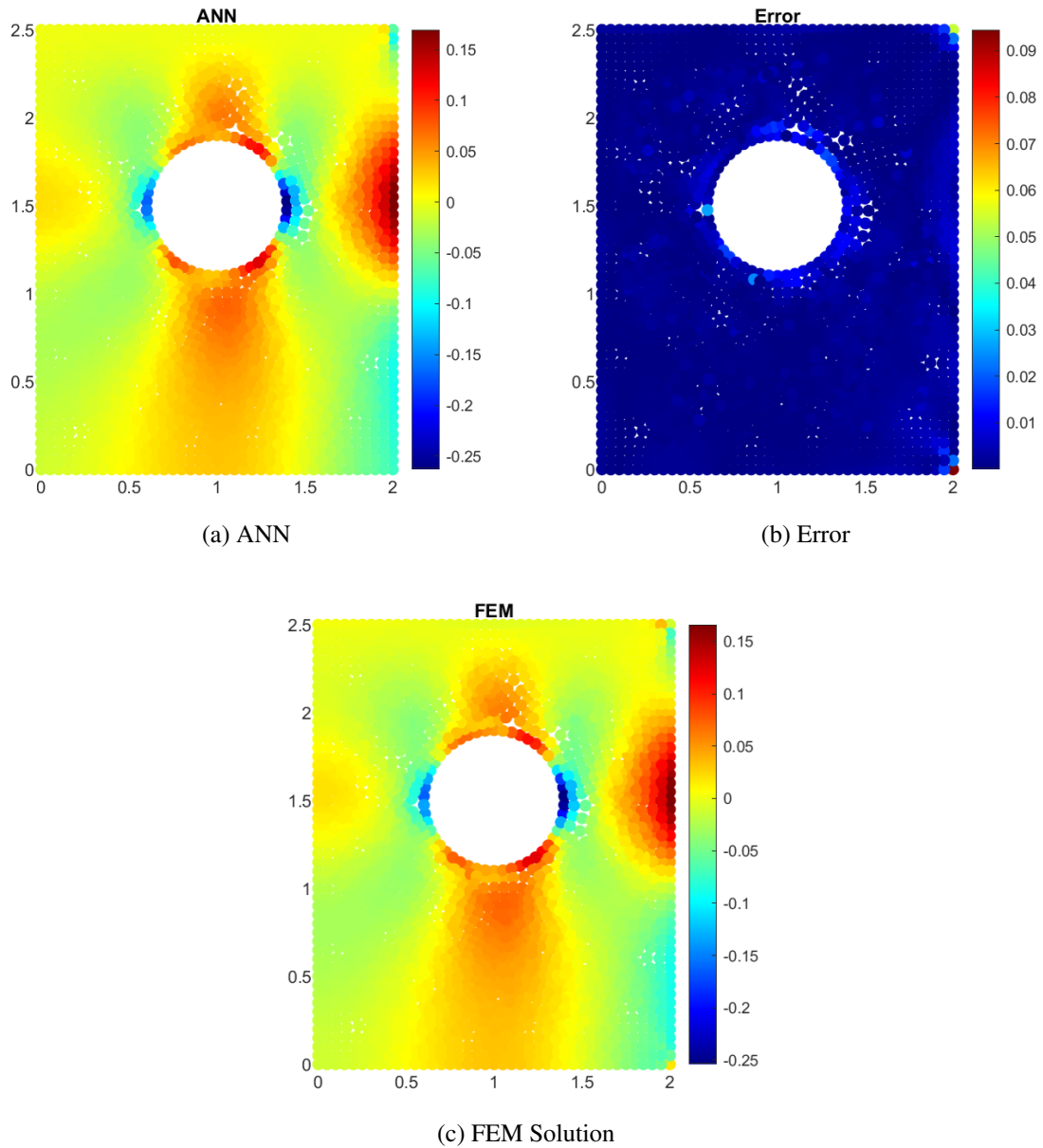


Figure 5.20: Predictions at the nodes for stress σ_{yy}

5.3.4.3 Stress τ_{xy}

The predictions for the shear stress τ_{xy} are presented at the below Figure 5.21.

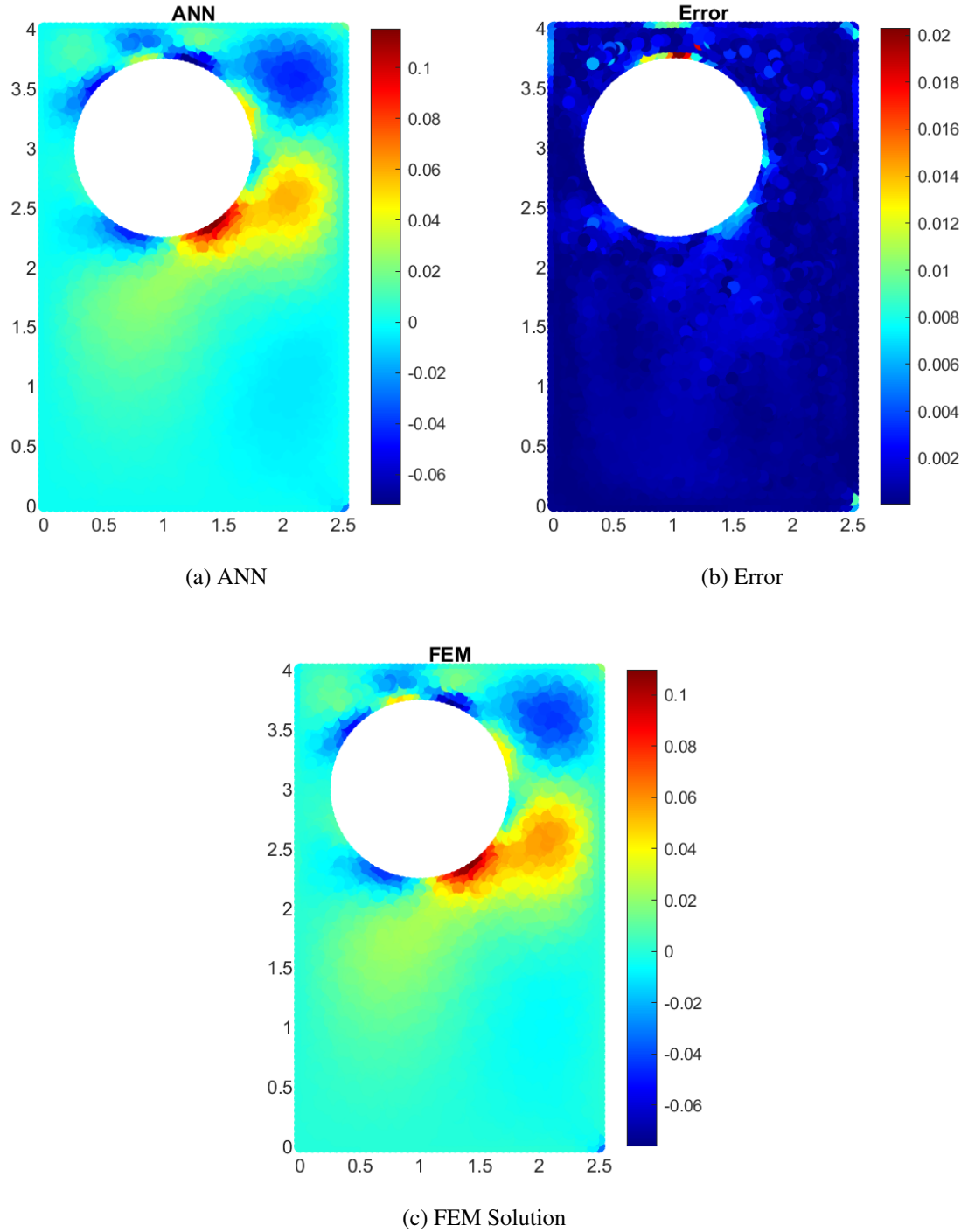


Figure 5.21: Predictions at the nodes for stress τ_{xy}

The error plot for all the stress components confirms that the neural network predicts these values with a high degree of accuracy.

5.3.5 Strain at Nodes

The trained network was also used to predict the strains at the nodes in this second case, employing the model that was originally trained on integration points. The results for each strain component are detailed in the sections below.

5.3.5.1 Strain ϵ_{xx}

Figure 5.22 shows the predictions made for the strains at the x direction. From the analysis of the figures, it can be extracted that the model effectively predicts ϵ_{xx} .

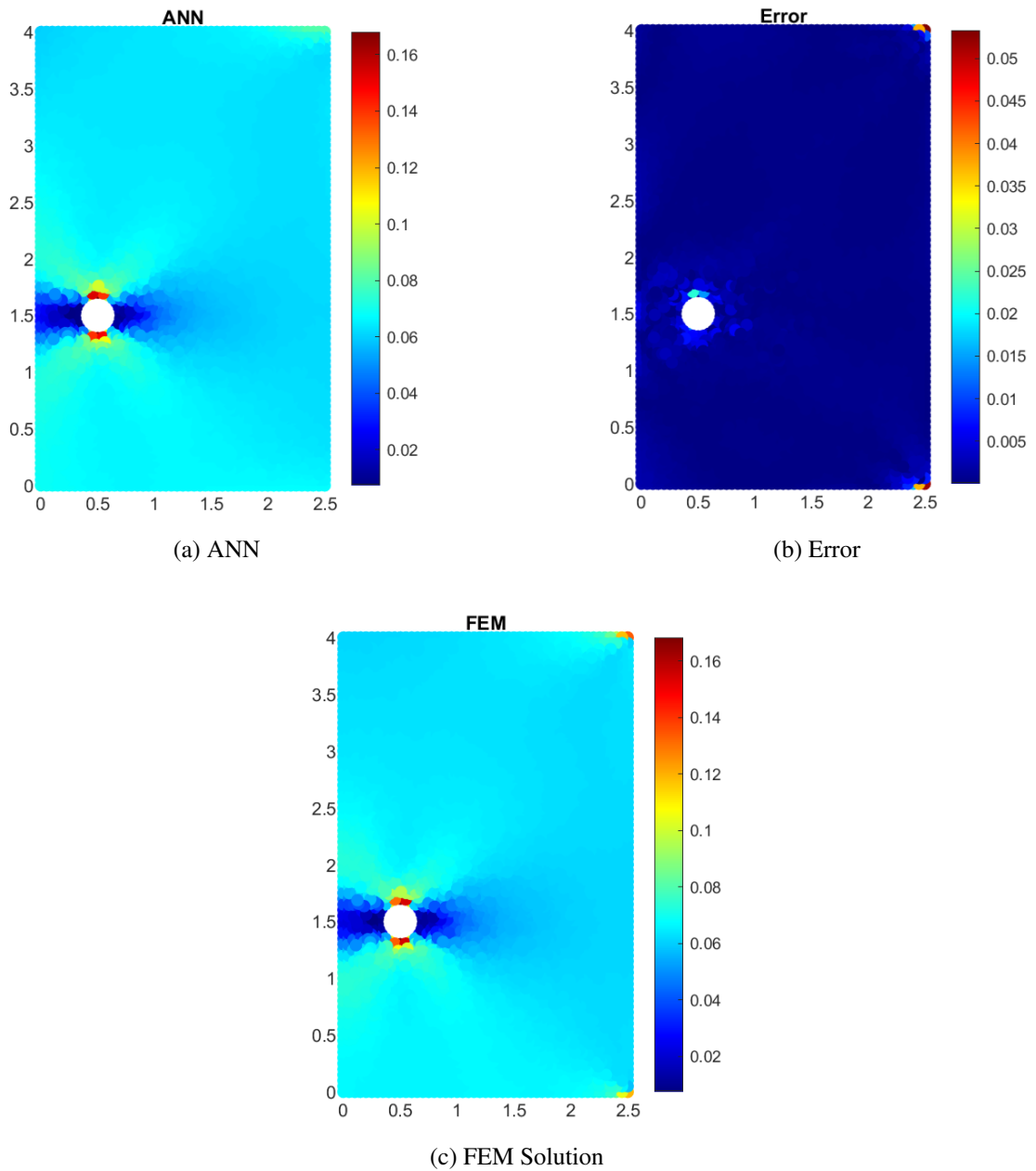


Figure 5.22: Predictions at the nodes for strain ϵ_{xx}

5.3.5.2 Strain ϵ_{yy}

For the strain in the y direction, ϵ_{yy} , the predictions are illustrated in Figure 5.23. As depicted, the ANN closely matches the true values, demonstrating its accuracy.

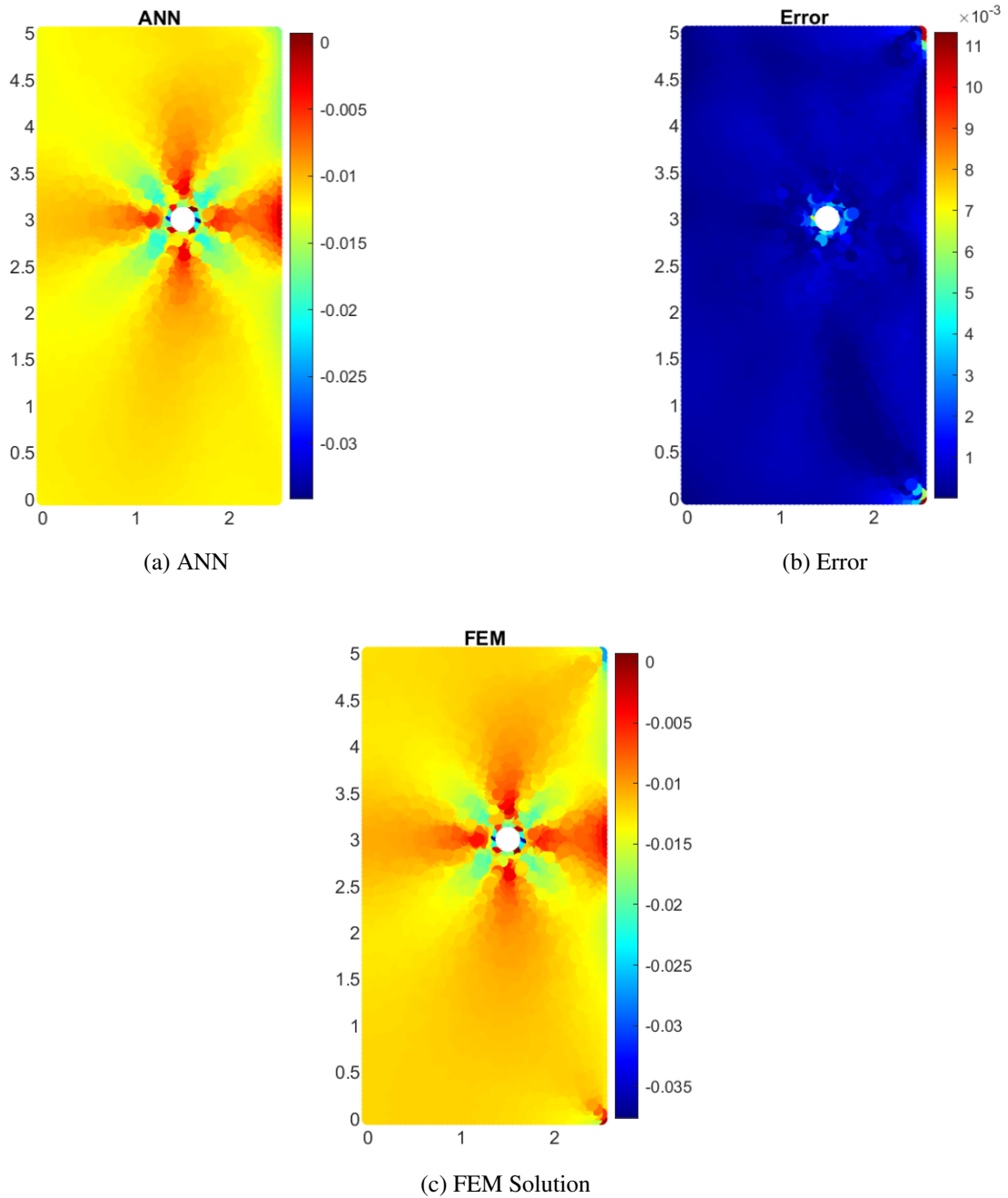


Figure 5.23: Predictions at the nodes for strain ϵ_{yy}

5.3.5.3 Strain ε_{xy}

The predictions for the shear strain on a specific plate with given dimensions are presented in Figure 5.24. Once again, the neural network delivers reliable predictions for this strain component.

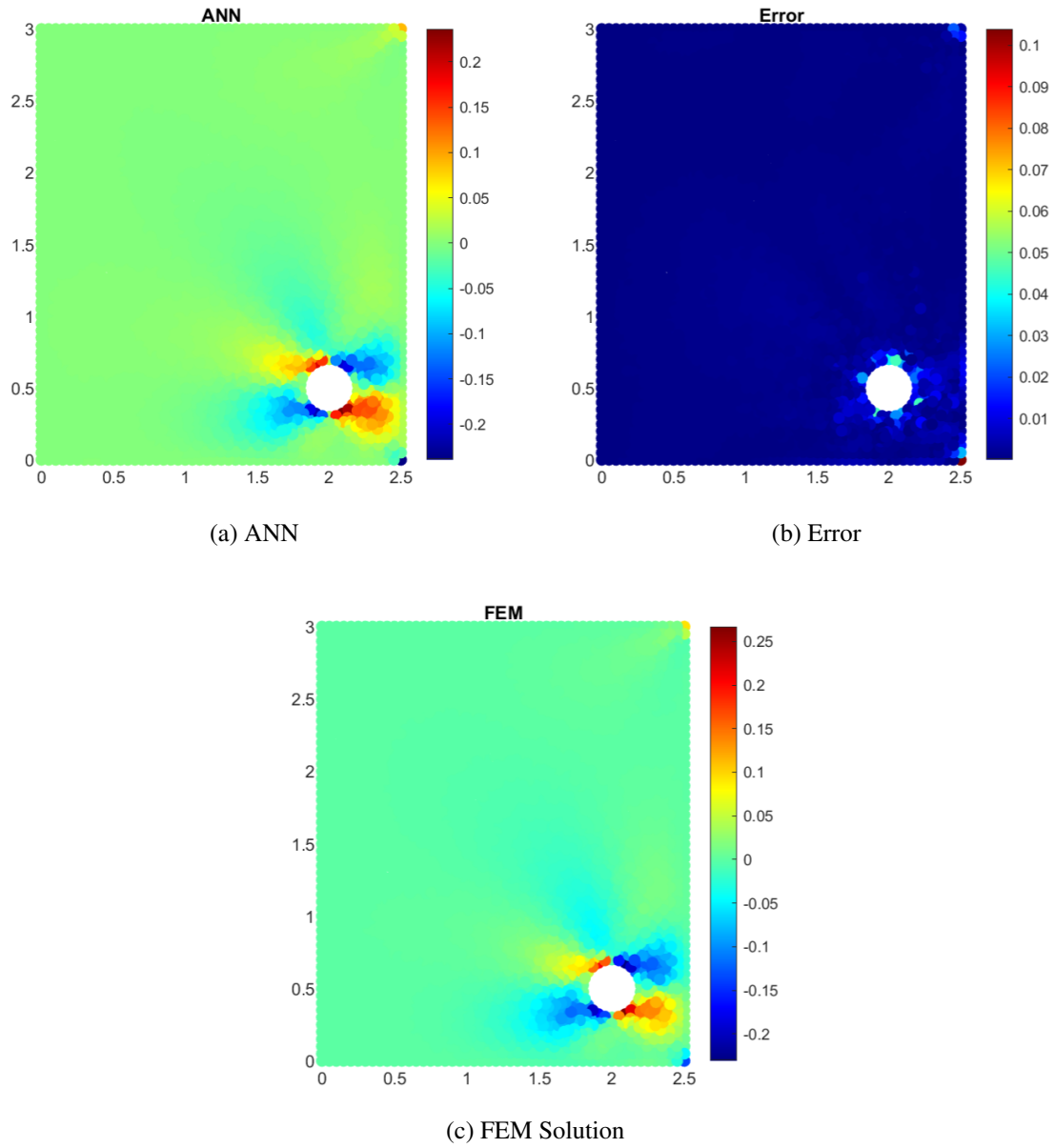


Figure 5.24: Predictions at the nodes for strain ε_{xy}

5.4 Discussion

For the second problem, involving a plate with four identical holes, the optimal set of hyperparameters identified in the first case was applied to evaluate their effectiveness in this new context. The goal was to determine whether these hyperparameters could generalize to a different scenario.

The neural network successfully predicted the displacements at the integration points. The model delivered robust and accurate predictions across different geometric configurations. Similar to the first case, the most significant errors were observed near the holes and at the top right corner of the plate, where stress concentrations are higher due to the geometric discontinuities.

For stress predictions, the number of training epochs was increased from 1000 to 2000. This decision was based on the fact that the results from the nodes showed that the model had the potential to improve further with additional training. The validation MAE across the epoch, as illustrated in Figures 5.7, 5.9 and 5.11, confirm that the extended training allowed the model to achieve a stable learning process, with significant improvements in performance over time. These graphs suggest that the chosen hyperparameters were effective.

Similar to the stress predictions, the neural network performed well when predicting strain components. The comparison with the FEM solutions demonstrates that the ANN model was able to accurately capture the strain distribution throughout the plate.

Overall, the hyperparameters chosen from the first problem proved to be well suited for this more complex case, suggesting that they have strong generalization capabilities. This result is significant, as it indicates that the same set of hyperparameters can be applied across different problems with similar characteristics, potentially reducing the need for extensive tuning in future cases.

In addition to predictions at the integration points, the neural network was also tasked with predicting stresses and strains at the nodes. The network demonstrated remarkable accuracy in these predictions, extending its capabilities beyond the integration points and showcasing its ability to effectively imitate the FEM nodal predictions.

Chapter 6

Conclusions

This research focused on developing an Artificial Neural Network (ANN) as a surrogate model for the Finite Element Method (FEM) in predicting the mechanical behaviour of well known Solid Mechanics benchmarks. The study began by analysing the two dimensional plate with a central hole subjected to a distributed load and exploring the impact that outliers have in the data, comparing the stability and accuracy of the ANN when trained with and without outliers. Moreover, the influence of the hyperparameters on the model performance was analysed and the optimal set of hyperparameters was obtained for each output.

A plate with four identical holes subjected to the same distributed load was also a case of study. The optimal set of hyperparameters previously identified for each output in the first case, was applied for the corresponding output in this case and their effectiveness was assessed by the performance of the model graph and the predictions made on unseen data.

From the first case, it was evident that the presence of outliers significantly impacts the accuracy and reliability of the artificial neural network. The network consistently produced better results across all outputs when trained on data without outliers, as opposed to when outliers were included. Regarding hyperparameters, the *tanh* activation function outperformed the *ReLU* function when outliers were excluded from the training data. Additionally, a learning rate of 0.0001 generally yielded more accurate results than a learning rate of 0.01.

For each output - node displacement, stress and strain - of the plate with a central hole, an optimal set of hyperparameters was identified, representing the configuration that produced the best results. The optimal set identified for the first case was then applied to the second case of a plate with four identical holes subjected to a distributed load in order to evaluate their effectiveness on the corresponding outputs.

The study also explored the potential of using a network trained on integration points to predict stress and strain components at the nodes. The results confirmed that the network, initially trained on integration points, was able to accurately predict those values at the nodes, demonstrating its versatility and effectiveness in extending predictions beyond its original training context.

The model of the plate with four identical holes performed with a high accuracy and minimal errors across all the outputs using the previously defined set of hyperparameters. With this, it is

possible to assume that these hyperparameters could be effectively applied to similar problems, indicating strong potential for generalisation.

For this case, the network also trained on integration points was used to predict stresses and strains at the nodes. Consistent with the results from the first case, the network exhibited a high degree of accuracy in these predictions. One of the key insights drawn from this analysis is the ability of the neural network's non-linear operations to effectively mimic the interpolation functions typically used in FEM for predicting nodal values.

For both of the analysed problems, the more pronounced errors are situated near the plate holes which shows that the model has slightly more difficulty in predicting the outputs in regions where the stress concentrations are higher.

Overall, this work demonstrates the viability of using Artificial Neural Networks as surrogate models for the Finite Element Method in solid mechanics. The findings emphasize the importance of careful data preparation and hyperparameter tuning in achieving reliable predictions. The successful generalisation of the model to different but related problems suggests that similar approaches could be extended to a broader range of applications in computational mechanics.

6.1 Study Limitations

The study limitations of this work are:

1. **Computation Resources:** The computational resources available for this study were constrained, leading to longer training times for each output. This limitation restricted the ability to explore a broader range of hyperparameters or more complex models
2. **Generality of Hyperparameters:** While the hyperparameters selected in this study yielded strong performance for the specific cases of a plate with a central hole and a plate with four holes, their applicability to other problems in Solid Mechanics remains uncertain. The results may not generalise well to different geometries, material properties or loading conditions.
3. **Data Quality and Outliers:** The process of identifying and removing the outliers on the data was manual, which could have introduced bias.
4. **Limited Exploration of Alternative Models:** This study only focused on the use of ANNs. While they proved to be effective, other machine learning models could have been used and analysed. The limited exploration of alternative models was influenced by time constraints.

6.2 Fulfillment of objectives

All the objectives defined in the beginning of the work were met and successfully achieved.

6.3 Future work

To enhance the research, the following ideas are suggested for future work:

1. **Expansion to Diverse Solid Mechanics Problems:** Extending the application of the ANNs to different Solid Mechanics problems using a wider variety of geometries, loading conditions and materials.
2. **Exploration of Alternative Machine Learning Techniques:** Explore and analyse different Machine Learning techniques to assess their performance.
3. **Enhanced Modeling of Stress Concentration Regions:** Improve the model performance on regions where the stress concentrations are higher.
4. **Automated Outlier Detection:** To improve data quality and reduce potential bias, future studies could incorporate automated outlier detection techniques.

References

- M. Atta, A. A. Abd-Elhady, A. Abu-Sinna, and H. E.M. Sallam. Prediction of failure stages for double lap joints using finite element analysis and artificial neural networks. *Engineering Failure Analysis*, 97(July 2018):242–257, 2019. ISSN 13506307. doi: 10.1016/j.engfailanal.2019.01.042. URL <https://doi.org/10.1016/j.engfailanal.2019.01.042>.
- Poojitha Vurtur Badarinath, Maria Chierichetti, and Fatemeh Davoudi Kakhki. A machine learning approach as a surrogate for a finite element analysis: Status of research and application to one dimensional systems. *Sensors*, 21(5):1–18, 2021. ISSN 14248220. doi: 10.3390/s21051654.
- Jorge Belinha. *Meshless Methods in Biomechanics: Bone Tissue Remodelling Analysis. Lecture Notes in Computational Vision and Biomechanics*. Springer, 2014. ISBN 3319064002. URL <https://books.google.com/books?id=0c4lBAAAQBAJ{&}pgis=1>.
- Anindya Bhaduri, Ashwini Gupta, and Lori Graham-Brady. Stress field prediction in fiber-reinforced composite materials using a deep learning approach. *Composites Part B: Engineering*, 238(April):109879, 2022. ISSN 13598368. doi: 10.1016/j.compositesb.2022.109879. URL <https://doi.org/10.1016/j.compositesb.2022.109879>.
- Hamed Bolandi, Xuyang Li, Talal Salem, Vishnu Naresh Boddeti, and Nizar Lajnef. Bridging finite element and deep learning: High-resolution stress distribution prediction in structural components. *Frontiers of Structural and Civil Engineering*, 16(11):1365–1377, 2022. ISSN 20952449. doi: 10.1007/s11709-022-0882-5.
- S. S. Quek G. R. Liu. *The Finite Element Method: A Practical Course*. Butterworth-Heinemann, 2003.
- Wenli Gao, Xinming Lu, Yanjun Peng, and Liang Wu. A Deep Learning Approach Replacing the Finite Difference Method for in Situ Stress Prediction. *IEEE Access*, 8:44063–44074, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2977880.
- A. Gholipour and N. Arjmand. Artificial neural networks to predict 3D spinal posture in reaching and lifting activities; Applications in biomechanical models. *Journal of Biomechanics*, 49(13): 2946–2952, 2016. ISSN 18732380. doi: 10.1016/j.jbiomech.2016.07.008. URL <http://dx.doi.org/10.1016/j.jbiomech.2016.07.008>.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2017. <http://www.deeplearningbook.org>.
- Education Harvard. Principle of virtual work and the Finite Element Method. *Harvard*, pages 1–15, 2007.

- Ali Hashemi, Jinwoo Jang, and Javad Beheshti. A Machine Learning-Based Surrogate Finite Element Model for Estimating Dynamic Response of Mechanical Systems. *IEEE Access*, 11 (May):54509–54525, 2023. ISSN 21693536. doi: 10.1109/ACCESS.2023.3282453.
- Cheng Hsi Hsiao, Albert Y. Chen, Louis Ge, and Fu Hsuan Yeh. Performance of artificial neural network and convolutional neural network on slope failure prediction using data from the random finite element method. *Acta Geotechnica*, 17(12):5801–5811, 2022. ISSN 18611133. doi: 10.1007/s11440-022-01520-w. URL <https://doi.org/10.1007/s11440-022-01520-w>.
- IBM. What is deep learning?, 2017. Available at <https://www.ibm.com/topics/deep-learning>.
- Haoliang Jiang, Zhenguo Nie, Roselyn Yeo, Amir Barati Farimani, and Levent Burak Kara. Stress-GAN: A generative deep learning model for two-dimensional stress distribution prediction. *Journal of Applied Mechanics, Transactions ASME*, 88(5):1–9, 2021. ISSN 15289036. doi: 10.1115/1.4049805.
- A. R. Khoei, H. Moslemi, and M. R. Seddighian. An efficient stress recovery technique in adaptive finite element method using artificial neural network. *Engineering Fracture Mechanics*, 237(April):107231, 2020. ISSN 00137944. doi: 10.1016/j.engfracmech.2020.107231. URL <https://doi.org/10.1016/j.engfracmech.2020.107231>.
- Miroslav Kubat. *An Introduction to Machine Learning*. Springer, 2017. ISBN 9783319639130. doi: 10.1007/978-3-319-63913-0.
- Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of the Royal Society Interface*, 15(138), 2018. ISSN 17425662. doi: 10.1098/rsif.2017.0844.
- Wing Kam Liu, Shaofan Li, and Harold S. Park. Eighty Years of the Finite Element Method: Birth, Evolution, and Future. *Archives of Computational Methods in Engineering*, 29(6):4431–4453, 2022. ISSN 18861784. doi: 10.1007/s11831-022-09740-9. URL <https://doi.org/10.1007/s11831-022-09740-9>.
- D. Lorente, F. Martínez-Martínez, M. J. Rupérez, M. A. Lago, M. Martínez-Sober, P. Escandell-Montero, J. M. Martínez-Martínez, S. Martínez-Sanchis, A. J. Serrano-López, C. Monserrat, and J. D. Martín-Guerrero. A framework for modelling the biomechanical behaviour of the human liver during breathing in real time using machine learning. *Expert Systems with Applications*, 71:342–357, 2017. ISSN 09574174. doi: 10.1016/j.eswa.2016.11.037. URL <http://dx.doi.org/10.1016/j.eswa.2016.11.037>.
- Alessandro De Luca, Marco Perfetto, Giuseppe Lamanna, and Francesco Caputo. Damage Detection in Flat Panels by Guided Waves Based. *Materials*, 2021.
- Montesinos López, Osva Antonio Montesinos López, Abelardo Crossa, and José. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Springer, 2022. ISBN 9783030890094. doi: 10.1007/978-3-030-89010-0.
- Ali Madani, Ahmed Bakhaty, Jiwon Kim, Yara Mubarak, and Mohammad R.K. Mofrad. Bridging Finite Element and Machine Learning Modeling: Stress Prediction of Arterial Walls in Atherosclerosis. *Journal of Biomechanical Engineering*, 141(8), 2019. ISSN 15288951. doi: 10.1115/1.4043290.

- Abdul Hamid Sheikh Madhujit Mukhopadhyay. *Matrix and Finite Element Analyses of Structures*. Springer, 2022.
- F. Martínez-Martínez, M. J. Rupérez-Moreno, M. Martínez-Sober, J. A. Solves-Llorens, D. Lorente, A. J. Serrano-López, S. Martínez-Sanchis, C. Monserrat, and J. D. Martín-Guerrero. A finite element-based machine learning approach for modeling the mechanical behavior of the breast tissues under compression in real-time. *Computers in Biology and Medicine*, 90 (September):116–124, 2017. ISSN 18790534. doi: 10.1016/j.compbiomed.2017.09.019.
- Siphesihle Mpho Motsa, Georgios Stavroulakis, and Georgios Drosopoulos. A data-driven, machine learning scheme used to predict the structural response of masonry arches. *Engineering Structures*, 296(July):116912, 2023. ISSN 18737323. doi: 10.1016/j.engstruct.2023.116912. URL <https://doi.org/10.1016/j.engstruct.2023.116912>.
- Saeed Mouloudi, Hadi Rahmanpanah, Colin Burvill, and Helen MS Davies. Prediction of displacement in the equine third metacarpal bone using a neural network prediction algorithm. *Biocybernetics and Biomedical Engineering*, 40(2):849–863, 2020. ISSN 02085216. doi: 10.1016/j.bbe.2019.09.001. URL <https://doi.org/10.1016/j.bbe.2019.09.001>.
- Rita Moura, Dulce A. Oliveira, João P.S. Ferreira, Marco P.L. Parente, Nina Kimmich, and Renato M. Natal Jorge. A finite element-based machine learning framework to predict the mechanical behavior of the pelvic floor muscles during childbirth. *Expert Systems with Applications*, 250(April):123953, 2024. ISSN 09574174. doi: 10.1016/j.eswa.2024.123953. URL <https://doi.org/10.1016/j.eswa.2024.123953>.
- Zhenguo Nie, Haoliang Jiang, and Levent Burak Kara. Stress field prediction in cantilevered structures using convolutional neural networks. *Journal of Computing and Information Science in Engineering*, 20(1):1–8, 2020. ISSN 15309827. doi: 10.1115/1.4044097.
- Murat Tolga Ozkan and Fulya Erdemir. Determination of theoretical stress concentration factor for circular/elliptical holes with reinforcement using analytical, finite element method and artificial neural network techniques. *Neural Computing and Applications*, 33(19):12641–12659, 2021. ISSN 14333058. doi: 10.1007/s00521-021-05914-x. URL <https://doi.org/10.1007/s00521-021-05914-x>.
- A. I. Pais, J. Belinha, and J. L. Alves. Advances in Computational Techniques for Bio-Inspired Cellular Materials in the Field of Biomechanics: Current Trends and Prospects. *Materials*, 16 (11), 2023a. ISSN 19961944. doi: 10.3390/ma16113946.
- Ana Pais, Jorge Lino Alves, and Jorge Belinha. Predicting trabecular arrangement in the proximal femur: An artificial neural network approach for varied geometries and load cases. *Journal of Biomechanics*, 161(November):111860, 2023b. ISSN 18732380. doi: 10.1016/j.jbiomech.2023.111860. URL <https://doi.org/10.1016/j.jbiomech.2023.111860>.
- J P A Ribeiro, L F F Gomes, and S M O Tavares. Artificial Neural Networks Applied in Mechanical Structural Design. *Journal of Computation and Artificial Intelligence in Mechanics and Biomechanics*, 1(1):14–21, 2021a. ISSN 2184-8971. doi: 10.5281/zenodo.4669797.
- João P.A. Ribeiro, Sérgio M.O. Tavares, and Marco Parente. Stress–strain evaluation of structural parts using artificial neural networks. *Proceedings of the Institution of Mechanical Engineers, Part L: Journal of Materials: Design and Applications*, 235(6):1271–1286, 2021b. ISSN 20413076. doi: 10.1177/1464420721992445.

- Bedanta B. Saikia, Dipjyoti Nath, and Sachin S. Gautam. Application of machine learning in efficient stress recovery in finite element analysis. *Materials Today: Proceedings*, 78:359–363, 2023. ISSN 22147853. doi: 10.1016/j.matpr.2022.10.100. URL <https://doi.org/10.1016/j.matpr.2022.10.100>.
- Erol Sancaktar. *Mechanics of solids and structures*, volume 10 PART A. Elsevier, 2008. ISBN 0791843041. doi: 10.1016/b978-0-12-818283-3.00002-6.
- Yasin Shokrollahi, Matthew M. Nikahd, Kimia Gholami, and Ghasem Azamirad. Deep Learning Techniques for Predicting Stress Fields in Composite Materials: A Superior Alternative to Finite Element Analysis. *Journal of Composites Science*, 7(8):1–12, 2023. ISSN 2504477X. doi: 10.3390/jcs7080311.
- M. Soleimani, B. Dashtbozorg, M. Mirkhalaf, and S. M. Mirkhalaf. A multiphysics-based artificial neural networks model for atherosclerosis. *Heliyon*, 9(7):e17902, 2023. ISSN 24058440. doi: 10.1016/j.heliyon.2023.e17902. URL <https://doi.org/10.1016/j.heliyon.2023.e17902>.
- Eduardo Soudah, José F. Rodríguez, and Roberto López. Mechanical stress in abdominal aortic aneurysms using artificial neural networks. *Journal of Mechanics in Medicine and Biology*, 15(3), 2015. ISSN 02195194. doi: 10.1142/S0219519415500293.
- J. Stanke, D. Trauth, A. Feuerhack, and F. Klocke. Setup of a Parameterized FE Model for the Die Roll Prediction in Fine Blanking using Artificial Neural Networks. *Journal of Physics: Conference Series*, 896(1), 2017. ISSN 17426596. doi: 10.1088/1742-6596/896/1/012096.
- Zhenya Sun, Zhenkun Lei, Jianchao Zou, Ruixiang Bai, Hao Jiang, and Cheng Yan. Prediction of failure behavior of composite hat-stiffened panels under in-plane shear using artificial neural network. *Composite Structures*, 272(June):114238, 2021. ISSN 02638223. doi: 10.1016/j.compstruct.2021.114238. URL <https://doi.org/10.1016/j.compstruct.2021.114238>.
- Sipho G. Thango, Georgios A. Drosopoulos, Siphesihle M. Motsa, and Georgios E. Stavroulakis. Prediction of the Response of Masonry Walls under Blast Loading Using Artificial Neural Networks. *Infrastructures*, 9(1), 2024. ISSN 24123811. doi: 10.3390/infrastructures9010005.
- S. Timoshenko and J.N. Goodier. *Theory of Elasticity*. McGraw-Hill Book Company, 2 edition, 1951.
- Yingying Wang, Yibin Li, Yong Song, and Xuewen Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences (Switzerland)*, 10(5), 2020. ISSN 20763417. doi: 10.3390/app10051897.

Appendix A

Appendix

A.1 Annex 1

```
1 import csv
2
3 # Function to calculate the smaller dimension of (x, y)
4 def calculate_smaller_dimension(x, y):
5     return min(x, y) - 0.1
6
7 # Open a CSV file in write mode
8 with open('output.csv', 'w', newline='') as csvfile:
9     # Create a CSV writer object
10    csv_writer = csv.writer(csvfile)
11
12    # Write header row
13    csv_writer.writerow(['x', 'y', 'r'])
14
15    # Initialize x, y with starting values
16    x = 1.0
17
18    # Loop until x reaches or exceeds 5.0
19    while x <= 5.0:
20        y = 1.0 # Reset y to 1.0 every time x increments
21        while y <= 5.0:
22            # Calculate the smaller dimension (r) of (x, y)
23            r = calculate_smaller_dimension(x, y)
24
25            # Iterate over r from 0.05 up to the smaller dimension of (x, y) - 0.1
26            r_step = 0.05
27            while r >= 0.0:
28                # Write the current values of x, y, and r to the CSV file
29                csv_writer.writerow([x, y, round(r, 2)]) # Round r to 2 decimal
30                places
31                r -= r_step # Decrement r by 0.05
```

```
31         y += 0.5 # Increment y by 0.5
32         x += 0.5 # Increment x by 0.5
```

A.2 Annex 2

```
1     from abaqus import *
2     from abaqusConstants import *
3     from odbAccess import *
4
5     from caeModules import *
6     from driverUtils import executeOnCaeStartup
7
8     import os
9     import csv
10    import numpy as np
11    import math
12
13    import mesh
14
15    import sys
16
17    filename = sys.argv[-1]
18
19    mesh_size = 0.05
20
21    folder_path = 'database'
22
23    # Check if the folder exists
24    if not os.path.exists(folder_path):
25        # Create the folder
26        os.makedirs(folder_path)
27
28
29
30    def replace_string_in_file(file_path, new_file_path=None):
31        # Define the original and replacement strings
32        original_string = "** FIELD OUTPUT: F-Output-1 \n ** \n *Output, field \n *Node
        Output \n CF, COORD, RF, U \n *Element Output, directions=YES \n IVOL, LE,
        PE, PEEQ, PEMAG, S \n *Contact Output \n CDISP, CSTRESS"
33
34        replacement_string = "** FIELD OUTPUT: F-Output-1\n **\n *Output, field\n *Node
        Output\n CF, RF, U\n *Element Output, directions=YES \n IVOL, LE, PE, PEEQ
        , PEMAG, S, COORD \n *Contact Output \n CDISP, CSTRESS"
35
36        # Open the original file
37        with open(file_path, 'r', encoding='utf-8') as file:
```

```

38     file_contents = file.read()
39
40     # Replace the original string with the replacement string
41     modified_contents = file_contents.replace(original_string, replacement_string)
42
43     # Write the modified contents to a new file or overwrite the original
44     output_path = new_file_path if new_file_path else file_path
45     with open(output_path, 'w', encoding='utf-8') as file:
46         file.write(modified_contents)
47
48
49
50
51 def get_nodes_on_edge(nodes,dim,x):
52     #n is the array of nodes in the part
53     #dim is if the edge is vertical or horizontal
54     #x is the value of the relevant coordinate of the edge i.e if its y = 2 this
55         means dim = 1 x = 2
56     labels = []
57     for n in nodes:
58         if n.coordinates[dim]==x:
59             labels.append(n.label)
60
61     #the returned labels is the tuple that is necessary to create a sequence from
62     labels
63
64     return tuple(labels)
65
66
67
68 with open(filename+'.csv') as file_obj:
69     os.chdir(folder_path)
70     reader_obj = csv.reader(file_obj) #each line of the file has length [index0],
71         height [index1], radius [index2]
72     plate_id=0
73     for row in reader_obj:
74         plate_id = plate_id+1
75         job_name = filename + '_' +str(plate_id)
76         plate_length = float(row[0])
77         plate_height = float(row[1])
78         hole_radius = float(row[2])
79
80         model_name = 'Model-'+str(plate_id)
81         job_name = 'job_'+model_name
82
83         #make sketch
84         s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize
85             =200.0)
86         s.Line(point1=(0.0, hole_radius), point2=(0.0, plate_height))
87         s.Line(point1=(0.0, plate_height), point2=(plate_length, plate_height))
88         s.Line(point1=(plate_length, plate_height), point2=(plate_length, 0.0))

```

```

83     s.Line(point1=(plate_length, 0.0), point2=(hole_radius, 0.0))
84
85     s.ArcByCenterEnds(center=(0.0, 0.0), point1=(0.0, hole_radius), point2=(
86         hole_radius, 0.0), direction=CLOCKWISE)
87
88     #create part
89     p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=TWO_D_PLANAR,
90         type=DEFORMABLE_BODY)
91     p.BaseShell(sketch=s)
92     s.unsetPrimaryObject()
93     del mdb.models['Model-1'].sketches['__profile__']
94
95     #material and section
96     mdb.models['Model-1'].Material(name='Material-1')
97     mdb.models['Model-1'].materials['Material-1'].Elastic(table=((1.0, 0.3), ))
98     mdb.models['Model-1'].HomogeneousSolidSection(name='Section-1', material='
99         Material-1', thickness=None)
100     p = mdb.models['Model-1'].parts['Part-1']
101
102     #section assignment
103     f = p.faces
104     faces = f.getSequenceFromMask(mask=('#1 ]', ), )
105     region = p.Set(faces=faces, name='Set-1')
106     p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
107         offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=
108         FROM_SECTION)
109
110     #assembly
111     a = mdb.models['Model-1'].rootAssembly
112     a.Instance(name='Part-1-1', part=p, dependent=ON)
113     mdb.models['Model-1'].StaticStep(name='Step-1', previous='Initial')
114     mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(variables
115         =('S', 'PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF', 'CSTRESS', 'CDISP'
116         , 'IVOL', 'COORD'))
117
118     #mesh part - loads are applied to the nodes and therefore need the creation
119     #of a node set
120     p.seedPart(size=mesh_size, deviationFactor=0.1, minSizeFactor=0.1)
121     f = p.faces
122     pickedRegions = f.getSequenceFromMask(mask=('#1 ]', ), )
123     p.setMeshControls(regions=pickedRegions, elemShape=QUAD, technique=
124         STRUCTURED)
125     elemType1 = mesh.ElemType(elemCode=CPS4, elemLibrary=STANDARD)
126     p.setElementType(elemTypes=(elemType1, ), regions=p.Set(faces=faces, name='
127         Set-1'))
128
129     p.generateMesh()

```

```

122     n = p.nodes
123
124     #x symmetry boundary condition
125     labels_for_x_sym = get_nodes_on_edge(n,0,0)
126     p.SetFromNodeLabels(nodeLabels=labels_for_x_sym,name='Set-X-sym')
127     region_x_sym = a.instances['Part-1-1'].sets['Set-X-sym']
128     mdb.models['Model-1'].XsymmBC(name='BC-1', createStepName='Step-1', region=
        region_x_sym, localCsys=None)
129
130
131     #y symmetry boundary condition
132     labels_for_y_sym = get_nodes_on_edge(n,1,0)
133     p.SetFromNodeLabels(nodeLabels=labels_for_y_sym,name='Set-Y-sym')
134     region_y_sym = a.instances['Part-1-1'].sets['Set-Y-sym']
135     mdb.models['Model-1'].YsymmBC(name='BC-2', createStepName='Step-1', region=
        region_y_sym, localCsys=None)
136
137     #natural boundary condition
138     labels_for_load = get_nodes_on_edge(n,0,plate_length)
139     p.SetFromNodeLabels(nodeLabels=labels_for_load, name='Set-load')
140
141     number_of_nodes = len(p.sets['Set-load'].nodes)
142     each_force = 1/float(number_of_nodes)
143
144     print >>sys.__stdout__, str(number_of_nodes)
145
146     region_load = a.instances['Part-1-1'].sets['Set-load']
147     mdb.models['Model-1'].ConcentratedForce(name='Load-1', createStepName='Step
        -1', region=region_load, cfl=each_force, distributionType=UNIFORM,
        field='', localCsys=None)
148
149     mdb.Job(name=job_name, model='Model-1', description='', type=ANALYSIS,
150     atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
151     memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
152     explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
153     modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
154     scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=1,
155     numGPUs=0)
156
157     mdb.jobs[job_name].writeInput(consistencyChecking=OFF)
158
159     replace_string_in_file(job_name+'.inp', new_file_path=None):
160
161     mdb.jobs[job_name].submit(consistencyChecking=OFF)
162
163     odb = openOdb(job_name+'.odb')
164
165     if not os.path.exists(job_name):
166         os.makedirs(job_name)

```

```

167         os.chdir(job_name)
168     else:
169         os.chdir(job_name)
170
171     lastFrame = odb.steps['Step-1'].frames[-1]
172     stress = lastFrame.fieldOutputs['S'].getSubset(position=INTEGRATION_POINT)
173     stress_values=stress.values
174     strain = lastFrame.fieldOutputs['E'].getSubset(position=INTEGRATION_POINT)
175     strain_values=strain.values
176     coord = lastFrame.fieldOutputs['COORD'].getSubset(position =
177         INTEGRATION_POINT)
178     coord_values = coord.values
179     stress_nodal = lastFrame.fieldOutputs['S'].getSubset(position =
180         ELEMENT_NODAL)
181     strain_nodal = lastFrame.fieldOutputs['E'].getSubset(position =
182         ELEMENT_NODAL)
183
184     displacement = lastFrame.fieldOutputs['U'].getSubset(position = NODAL)
185     displacement_values = displacement.values
186
187     for v,w,x in zip(coord_values, stress_values, strain_values):
188         output = open(job_name+'_GP_data.csv','a')
189         output.write(str(v.data[0])+','+str(v.data[1])+','+str(w.data[0])+','+
190             str(w.data[1])+','+str(w.data[3])+','+str(x.data[0])+','+str(x.data
191             [1])+','+str(x.data[3])+','+str(plate_length)+'/'+str(plate_height)
192             +','+str(hole_radius)+'\n')
193         output.close()
194
195     displacement_array = []
196     for w,node in zip(displacement_values,n):
197         output = open(job_name+'_displacements_w_coords.csv','a')
198         output.write(str(node.label)+'/'+str(node.coordinates[0])+','+str(node.
199             coordinates[1])+','+str(w.nodeLabel)+'/'+str(w.data[0])+','+str(w.
200             data[1])+','+str(plate_length)+'/'+str(plate_height)+'/'+str(
201             hole_radius)+'\n')
202         output.close()
203
204     stress_nodes_array = []
205     stress_nodes_id = []
206     strain_nodes_array=[]
207     strain_nodes_id=[]
208     for value1,value2 in zip(stress_nodal.values,strain_nodal.values):
209         stress_nodes_array.append([value1.data[0],value1.data[1],value1.data
210             [3]])
211         stress_nodes_id.append(value1.nodeLabel)

```

```

205         strain_nodes_array.append([value2.data[0],value2.data[1],value2.data
206             [3]])
207         strain_nodes_id.append(value2.nodeLabel)
208
209     largest_node_label = max(stress_nodes_id)
210
211     average_nodal_stresses = []
212     average_nodal_strains = []
213     c=0
214     stress_xx =0
215     stress_yy =0
216     stress_xy =0
217     strain_xx =0
218     strain_yy =0
219     strain_xy =0
220
221     for i in range(1,largest_node_label+1):
222         for j in range(len(stress_nodes_id)):
223             if stress_nodes_id[j]==i:
224                 c = c+1
225                 stress_xx = stress_xx + stress_nodes_array[j][0]
226                 stress_yy = stress_yy + stress_nodes_array[j][1]
227                 stress_xy = stress_xy + stress_nodes_array[j][2]
228                 strain_xx = strain_xx + strain_nodes_array[j][0]
229                 strain_yy = strain_yy + strain_nodes_array[j][1]
230                 strain_xy = strain_xy + strain_nodes_array[j][2]
231
232     stress_xx = stress_xx/c
233     stress_yy = stress_yy/c
234     stress_xy = stress_xy/c
235     strain_xx = strain_xx/c
236     strain_yy = strain_yy/c
237     strain_xy = strain_xy/c
238
239     average_nodal_stresses.append([float(i),stress_xx,stress_yy,stress_xy])
240     average_nodal_strains.append([float(i),strain_xx,strain_yy,strain_xy])
241     c=0
242     stress_xx =0
243     stress_yy =0
244     stress_xy =0
245     strain_xx =0
246     strain_yy =0
247     strain_xy =0
248
249     for i in range(len(average_nodal_stresses)):
250         output = open(job_name+'_nodal_stresses_strains.csv','a')
251         output.write(str(average_nodal_stresses[i][0])+','+str(
252             average_nodal_stresses[i][1])+','+str(average_nodal_stresses[i][2])
253             +','+str(average_nodal_stresses[i][3])+','+str(
254             average_nodal_strains[i][0])+','+str(average_nodal_strains[i][1])+
255             ','+str(average_nodal_strains[i][2])+','+str(average_nodal_strains[i]

```



```

        ] [3]))+', '+str(plate_length)+' ', '+str(plate_height)+' ', '+str(
        hole_radius)+'\n')
249     output.close()
250
251     os.chdir('..')

```

A.3 Annex 3

```

1  import numpy as np
2  import pandas as pd
3  import tensorflow as tf
4  from sklearn.preprocessing import MinMaxScaler
5  import os
6  from sklearn.model_selection import KFold
7  import matplotlib.pyplot as plt
8  import keras
9  from sklearn.model_selection import train_test_split
10 import csv
11
12 from sklearn.metrics import accuracy_score
13
14
15 #Function to save model details
16 def save_model_details(model, filename):
17     with open(filename, 'w') as f:
18         # Print the model summary
19         model.summary(print_fn=lambda x: f.write(x + '\n'))
20         # Access and print the learning rate from the optimizer
21         learning_rate = float(tf.keras.backend.get_value(model.optimizer.
22             learning_rate))
23         f.write(f'Learning Rate: {learning_rate}\n')
24
25 # Function to load and process data
26 def load_and_process_data(ids):
27     data = []
28     for idx in ids:
29         dir_path = f'job_Model-{idx+1}'
30         file_path = os.path.join(dir_path, f'job_Model-{idx+1}_GP_data.csv')
31
32         if os.path.exists(dir_path):
33             if os.path.exists(file_path):
34                 temp_data = pd.read_csv(file_path, header=None)
35                 # Check the condition for the entire file
36                 if not ((temp_data.iloc[:, 10] / temp_data.iloc[:, 8] > 0.75).any()
37                     or (temp_data.iloc[:, 10] / temp_data.iloc[:, 9] > 0.75).any()
38                     ):

```

```

36         data.append(temp_data)
37     else:
38         print(f"File for job_Model-{idx+1} does not meet the condition.
39             Skipping...")
40
41     else:
42         print(f"File for job_Model-{idx+1} does not exist. Skipping...")
43     else:
44         print(f"Directory for job_Model-{idx+1} does not exist. Skipping...")
45
46     if data:
47         return pd.concat(data, ignore_index=True)
48     else:
49         print("No data loaded. Returning empty DataFrame.")
50         return pd.DataFrame()
51
52 #Function to save model
53 def create_model(layers, neurons, rate):
54     model = tf.keras.Sequential()
55     model.add(tf.keras.layers.InputLayer(input_shape=(5,)))
56     for _ in range(layers):
57         model.add(tf.keras.layers.Dense(neurons, activation='tanh'))
58     model.add(tf.keras.layers.Dense(5, activation = 'tanh'))
59     model.add(tf.keras.layers.Dense(1, activation = 'linear'))
60     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=rate), loss='mae
61         ')
62
63     return model
64
65 # GPU setup
66 gpus = tf.config.list_physical_devices('GPU')
67 if gpus:
68     try:
69         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
70         tf.config.experimental.set_memory_growth(gpus[0], True)
71         print(f"Using GPU: {gpus[0].name}")
72     except RuntimeError as e:
73         print(e)
74 else:
75     print("No GPU found. TensorFlow will use CPU.")
76
77 #Import Data
78 model_data = pd.read_csv('output.csv', header=None)
79
80 # Split data into training and testing
81 np.random.seed(42)
82 train_ids = np.sort(np.random.choice(model_data.index, size=int(0.7*len(model_data)
83     ), replace=False))

```

```
82 test_ids = np.setdiff1d(model_data.index, train_ids)
83
84 # Load training and testing data
85 training_data = load_and_process_data(train_ids)
86 testing_data = load_and_process_data(test_ids)
87
88 print("Data Imported")
89
90 feature_columns = [0, 1, 8, 9, 10]
91
92 #Normalize the data
93 scaler_features = MinMaxScaler()
94 scaler_targets = MinMaxScaler()
95
96 training_features = scaler_features.fit_transform(training_data.iloc[:,
    feature_columns].values)
97 training_targets = scaler_targets.fit_transform(training_data.iloc[:, 2].values.
    reshape(-1, 1)).flatten()
98 testing_features = scaler_features.transform(testing_data.iloc[:, feature_columns].
    values)
99 testing_targets = scaler_targets.transform(testing_data.iloc[:, 2].values.reshape
    (-1, 1)).flatten()
100
101 #Test if it is correct normalized
102 print("Training features range:", training_features.min(), training_features.max())
103 print("Training targets range:", training_targets.min(), training_targets.max())
104
105 print("Testing features range:", testing_features.min(), testing_features.max())
106 print("Testing targets range:", testing_targets.min(), testing_targets.max())
107
108
109 # Neural network settings
110 num_hidden_layers_options = [10, 20, 30 ]
111 neurons_per_layer_options = [8, 16, 32 ]
112 learning_rates_options = [0.01, 0.0001 ]
113 epochs_options = [1000]
114 batch_size_options = [32768, 65536]
115
116
117 #K-Fold Cross Validation
118 kf = KFold(n_splits = 3, shuffle = True, random_state = 42)
119 print("Folds are defined")
120
121 best_model = None
122 best_performance = np. inf
123
124 all_mae_histories = []
125 fold_no = 0
126
```

```

127 for train_index, test_index in kf.split(training_data):
128     fold_no += 1
129     print(f"Training for fold {fold_no}")
130
131     #Training
132     train_features = training_features[train_index]
133     train_targets = training_targets[train_index]
134     val_features = training_features[test_index]
135     val_targets = training_targets[test_index]
136
137
138     for num_layers in num_hidden_layers_options:
139         for neurons in neurons_per_layer_options:
140             for rate in learning_rates_options:
141                 for epochs in epochs_options:
142                     for batch_size in batch_size_options:
143                         print(f"num layers {num_layers}")
144                         print(f"neurons {neurons}")
145                         print(f"learning rate {rate}")
146                         print(f"num epochs {epochs}")
147                         print(f"batch size: {batch_size}")
148
149                         model = create_model(num_layers, neurons, rate)
150
151                         #Add EarlyStopping
152                         early_stopping = tf.keras.callbacks.EarlyStopping(monitor='
                            loss', min_delta=1.0e-8, patience=100, mode=min,
                            restore_best_weights=False)
153
154                         #Train the Model
155                         history = model.fit(train_features, train_targets, epochs =
                            epochs, batch_size = batch_size, verbose=2,
                            validation_data = (val_features, val_targets),
                            callbacks = [early_stopping])
156
157                         #Save Performance
158                         mae_data = pd.DataFrame({
159                             'Epoch': range(1, len(history.history['loss']) + 1),
160                             'Training_MAE': history.history['loss'],
161                             'Validation_MAE': history.history['val_loss']})
162
163                         mae_data.to_csv(f'{rate}_{num_layers}_{neurons}_{batch_size}
                            _mae_vs_epoch.csv', index=False)
164                         mae_history = history.history['val_loss']
165                         all_mae_histories.append(mae_history)
166                         loss = model.evaluate(val_features, val_targets, verbose =
                            2)
167
168                         #Check the Best Model

```

```

169             if loss < best_performance:
170                 best_model = model
171                 best_performance = loss
172                 best_history = mae_history
173
174 #Save the Best Model
175 best_model.save('best_model.h5')
176
177 # Average MSE history for the best model
178 plt.figure(figsize=(10, 6))
179 plt.plot(range(1, len(best_history) + 1), best_history)
180 plt.title('Best Model Validation MAE vs. Epochs')
181 plt.xlabel('Epochs')
182 plt.ylabel('Validation MAE')
183 plt.grid(True)
184 plt.savefig('best_model_mae.png')
185 plt.close()
186
187 #Predictions
188 predictions = best_model.predict(testing_features)
189 predictions = scaler_targets.inverse_transform(predictions).flatten()
190 true_values = scaler_targets.inverse_transform(testing_targets.reshape(-1, 1)).
    flatten()
191
192 predictions_df = pd.DataFrame({'True Values': true_values, 'Predictions':
    predictions})
193 predictions_df.to_csv('predictions.csv', index=False)
194
195 tf.keras.models.save_model(best_model, 'central_hole_net_stress_xx', save_format='
    tf')

```

A.4 Annex 4

```

1     import numpy as np
2     import pandas as pd
3     import tensorflow as tf
4     from sklearn.preprocessing import MinMaxScaler
5     import os
6     from sklearn.model_selection import KFold
7     import matplotlib.pyplot as plt
8     import keras
9     from sklearn.model_selection import train_test_split
10    import csv
11
12    from sklearn.metrics import accuracy_score
13

```

```

14
15 #Function to save model details
16 def save_model_details(model, filename):
17     with open(filename, 'w') as f:
18         # Print the model summary
19         model.summary(print_fn=lambda x: f.write(x + '\n'))
20         # Access and print the learning rate from the optimizer
21         learning_rate = float(tf.keras.backend.get_value(model.optimizer.
22             learning_rate))
23         f.write(f'Learning Rate: {learning_rate}\n')
24
25 # Function to load and process data
26 def load_and_process_data(ids):
27     data = []
28     for idx in ids:
29         dir_path = f'job_Model-{idx+1}'
30         file_path = os.path.join(dir_path, f'job_Model-{idx+1}_GP_data.csv')
31
32         if os.path.exists(dir_path):
33             if os.path.exists(file_path):
34                 temp_data = pd.read_csv(file_path, header=None)
35                 data.append(temp_data)
36             else:
37                 print(f"File for job_Model-{idx+1} does not exist. Skipping...")
38         else:
39             print(f"Directory for job_Model-{idx+1} does not exist. Skipping...")
40
41     if data:
42         return pd.concat(data, ignore_index=True)
43     else:
44         print("No data loaded. Returning empty DataFrame.")
45         return pd.DataFrame()
46
47 #Function to save model
48 def create_model(layers, neurons, rate):
49     model = tf.keras.Sequential()
50     model.add(tf.keras.layers.InputLayer(input_shape=(5,)))
51     for _ in range(layers):
52         model.add(tf.keras.layers.Dense(neurons, activation='tanh'))
53     model.add(tf.keras.layers.Dense(5, activation = 'tanh'))
54     model.add(tf.keras.layers.Dense(1, activation = 'linear'))
55     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=rate), loss='mae')
56
57     return model
58
59 # GPU setup
60 gpus = tf.config.list_physical_devices('GPU')
61 if gpus:

```

```

61     try:
62         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
63         tf.config.experimental.set_memory_growth(gpus[0], True)
64         print(f"Using GPU: {gpus[0].name}")
65     except RuntimeError as e:
66         print(e)
67 else:
68     print("No GPU found. TensorFlow will use CPU.")
69
70
71 #Import Data
72 model_data = pd.read_csv('output.csv', header=None)
73
74 # Split data into training and testing
75 np.random.seed(42)
76 train_ids = np.sort(np.random.choice(model_data.index, size=int(0.7*len(model_data)
77     ), replace=False))
78
79 test_ids = np.setdiff1d(model_data.index, train_ids)
80
81 # Load training and testing data
82 training_data = load_and_process_data(train_ids)
83 testing_data = load_and_process_data(test_ids)
84
85 print("Data Imported")
86
87 feature_columns = [0, 1, 8, 9, 10]
88
89 #Normalize the data
90 scaler_features = MinMaxScaler()
91 scaler_targets = MinMaxScaler()
92
93 training_features = scaler_features.fit_transform(training_data.iloc[:,
94     feature_columns].values)
95
96 training_targets = scaler_targets.fit_transform(training_data.iloc[:, 2].values.
97     reshape(-1, 1)).flatten()
98
99 testing_features = scaler_features.transform(testing_data.iloc[:, feature_columns].
100     values)
101
102 testing_targets = scaler_targets.transform(testing_data.iloc[:, 2].values.reshape
103     (-1, 1)).flatten()
104
105 #Test if it is correct normalized
106 print("Training features range:", training_features.min(), training_features.max())
107 print("Training targets range:", training_targets.min(), training_targets.max())
108
109 print("Testing features range:", testing_features.min(), testing_features.max())
110 print("Testing targets range:", testing_targets.min(), testing_targets.max())
111
112 # Neural network settings

```

```

105 num_hidden_layers_options = [10, 20, 30 ]
106 neurons_per_layer_options = [8, 16, 32 ]
107 learning_rates_options = [0.01, 0.0001 ]
108 epochs_options = [1000]
109 batch_size_options = [32768, 65536]
110
111
112 #K-Fold Cross Validation
113 kf = KFold(n_splits = 3, shuffle = True, random_state = 42)
114 print("Folds are defined")
115
116 best_model = None
117 best_performance = np. inf
118
119 all_mae_histories = []
120 fold_no = 0
121
122 for train_index, test_index in kf.split(training_data):
123     fold_no += 1
124     print(f"Training for fold {fold_no}")
125
126     #Training
127     train_features = training_features[train_index]
128     train_targets = training_targets[train_index]
129     val_features = training_features[test_index]
130     val_targets = training_targets[test_index]
131
132
133     for num_layers in num_hidden_layers_options:
134         for neurons in neurons_per_layer_options:
135             for rate in learning_rates_options:
136                 for epochs in epochs_options:
137                     for batch_size in batch_size_options:
138                         print(f"num layers {num_layers}")
139                         print(f"neurons {neurons}")
140                         print(f"learning rate {rate}")
141                         print(f"num epochs {epochs}")
142                         print(f"batch size: {batch_size}")
143
144                         model = create_model(num_layers, neurons, rate)
145
146                         #Add EarlyStopping
147                         early_stopping = tf.keras.callbacks.EarlyStopping(monitor='
                            loss', min_delta=1.0e-8, patience=100, mode=min,
                            restore_best_weights=False)
148
149                         #Train the Model
150                         history = model.fit(train_features, train_targets, epochs =
                            epochs, batch_size = batch_size, verbose=2,

```



```

validation_data = (val_features, val_targets),
callbacks = [early_stopping])

151
152         #Save Performance
153         mae_data = pd.DataFrame({
154             'Epoch': range(1, len(history.history['loss']) + 1),
155             'Training_MAE': history.history['loss'],
156             'Validation_MAE': history.history['val_loss'])
157
158         mae_data.to_csv(f'{rate}_{num_layers}_{neurons}_{batch_size}
                        }_mae_vs_epoch.csv', index=False)
159         mae_history = history.history['val_loss']
160         all_mae_histories.append(mae_history)
161         loss = model.evaluate(val_features, val_targets, verbose =
                        2)

162
163         #Check the Best Model
164         if loss < best_performance:
165             best_model = model
166             best_performance = loss
167             best_history = mae_history
168
169     #Save the Best Model
170     best_model.save('best_model.h5')
171
172     # Average MSE history for the best model
173     plt.figure(figsize=(10, 6))
174     plt.plot(range(1, len(best_history) + 1), best_history)
175     plt.title('Best Model Validation MAE vs. Epochs')
176     plt.xlabel('Epochs')
177     plt.ylabel('Validation MAE')
178     plt.grid(True)
179     plt.savefig('best_model_mae.png')
180     plt.close()
181
182     #Predictions
183     predictions = best_model.predict(testing_features)
184     predictions = scaler_targets.inverse_transform(predictions).flatten()
185     true_values = scaler_targets.inverse_transform(testing_targets.reshape(-1, 1)).
        flatten()
186
187     predictions_df = pd.DataFrame({'True Values': true_values, 'Predictions':
        predictions})
188     predictions_df.to_csv('predictions.csv', index=False)
189
190     tf.keras.models.save_model(best_model, 'central_hole_net_stress_xx', save_format='
        tf')

```

A.5 Annex 5

```

1      clc;
2  clear all;
3  close all;
4
5  % Load data from CSV files
6  inputs = readtable("testing_features.csv"); % Load the testing features
7  predictions_data = readtable("predictions.csv"); % Load the true values and
      predictions
8
9  % Extract targets and predictions
10 targets = predictions_data(:, 1); % First column: True Values
11 predictions = predictions_data(:, 2); % Second column: Predictions
12
13 % Extract unique combinations of features (columns 3, 4, 5)
14 b = inputs(:, 3:5);
15 unique_combinations = unique(b, 'rows', 'stable');
16
17 % Initialize Mean Absolute Error array
18 MAE = [];
19
20 % Create directory for figures if it doesn't exist
21 if ~exist('figs_u', 'dir')
22     mkdir('figs_u')
23 end
24 cd('figs_u')
25
26 % Loop over each unique combination to generate figures
27 for i = 1:length(unique_combinations)
28     idx = find(ismember(b, unique_combinations(i,:), 'rows'));
29     mae_i = abs(targets(idx) - predictions(idx));
30
31     % Get plate dimensions
32     x_coords = inputs{idx, 1};
33     y_coords = inputs{idx, 2};
34     x_min = min(x_coords);
35     x_max = max(x_coords);
36     y_min = min(y_coords);
37     y_max = max(y_coords);
38
39     % Plot FEM
40     figure;
41     scatter(x_coords, y_coords, 50, targets(idx), 'filled');
42     axis equal;
43     xlim([x_min, x_max]);
44     ylim([y_min, y_max]);
45     title('FEM');

```

```

46     colormap jet;
47     colorbar;
48     saveas(gcf, [num2str(i), '_disp_u_FEM.png']);
49     close;
50
51     % Plot ANN
52     figure;
53     scatter(x_coords, y_coords, 50, predictions(idx), 'filled');
54     axis equal;
55     xlim([x_min, x_max]);
56     ylim([y_min, y_max]);
57     title('ANN');
58     colormap jet;
59     colorbar;
60     saveas(gcf, [num2str(i), '_disp_u_ANN.png']);
61     close;
62
63     % Plot Error
64     figure;
65     scatter(x_coords, y_coords, 50, mae_i, 'filled');
66     axis equal;
67     xlim([x_min, x_max]);
68     ylim([y_min, y_max]);
69     title('Error');
70     colormap jet;
71     colorbar;
72     saveas(gcf, [num2str(i), '_disp_u_error.png']);
73     close;
74
75     % Calculate mean absolute error for the current combination
76     MAE(i) = mean(mae_i);
77 end
78
79 % Plot histogram of MAE
80 figure;
81 histogram(MAE);
82 title('Histogram of Mean Absolute Errors');
83 saveas(gcf, 'histogram_disp_u.png');
84 cd('..');

```

A.6 Annex 6

```

1     import numpy as np
2     import pandas as pd
3     import tensorflow as tf
4     from sklearn.preprocessing import MinMaxScaler

```

```

5 import os
6 from sklearn.model_selection import KFold
7 import matplotlib.pyplot as plt
8 import keras
9 from sklearn.model_selection import train_test_split
10 import csv
11
12 from sklearn.metrics import accuracy_score
13
14 # Function to load and process data
15 def load_and_process_data(ids):
16     data = []
17     for idx in ids:
18         dir_path = f'job_Model-{idx+1}'
19         file_path = os.path.join(dir_path, f'job_Model-{idx+1}_GP_data.csv')
20
21         if os.path.exists(dir_path):
22             if os.path.exists(file_path):
23                 temp_data = pd.read_csv(file_path, header=None)
24                 data.append(temp_data)
25             else:
26                 print(f"File for job_Model-{idx+1} does not exist. Skipping...")
27         else:
28             print(f"Directory for job_Model-{idx+1} does not exist. Skipping...")
29
30     if data:
31         return pd.concat(data, ignore_index=True)
32     else:
33         print("No data loaded. Returning empty DataFrame.")
34         return pd.DataFrame()
35
36 # Function to load and process data
37 def load_and_process_data2(ids):
38     data = []
39     for idx in ids:
40         dir_path = f'job_Model-{idx+1}'
41         file_path1 = os.path.join(dir_path, f'job_Model-{idx+1}_nodal_stresses_strains.csv')
42         file_path2 = os.path.join(dir_path, f'job_Model-{idx+1}_displacements_w_coords.csv')
43
44
45         if os.path.exists(dir_path):
46             if os.path.exists(file_path1) and os.path.exists(file_path2):
47                 temp_data1 = pd.read_csv(file_path1, header=None)
48                 temp_data2 = pd.read_csv(file_path2, header=None)
49
50                 combined_data = pd.concat([temp_data2.iloc[:, [1, 2]], temp_data1.
                    iloc[:, [1,2,3,5,6,7,8,9,10,11,12]]], axis=1)

```

```

51         data.append(combined_data)
52     else:
53         print(f"File for job_Model-{idx+1} does not exist. Skipping...")
54     else:
55         print(f"Directory for job_Model-{idx+1} does not exist. Skipping...")
56
57     if data:
58         return pd.concat(data, ignore_index=True)
59     else:
60         print("No data loaded. Returning empty DataFrame.")
61         return pd.DataFrame()
62
63
64 #Import Data
65 model_data = pd.read_csv('output.csv', header=None)
66
67 # Split data into training and testing
68 np.random.seed(42)
69 train_ids = np.sort(np.random.choice(model_data.index, size=int(0.7*len(model_data)
70                                     ), replace=False))
71 test_ids = np.setdiff1d(model_data.index, train_ids)
72
73 # Load training and testing data
74 training_data = load_and_process_data(train_ids)
75 testing_data = load_and_process_data2(test_ids)
76
77 print("Data Imported")
78
79 feature_columns = [0, 1, 8, 9, 10, 11, 12]
80
81 #Normalize the data
82 scaler_features = MinMaxScaler()
83 scaler_targets = MinMaxScaler()
84
85 training_features = scaler_features.fit_transform(training_data.iloc[:,
86                                                    feature_columns].values)
87 training_targets = scaler_targets.fit_transform(training_data.iloc[:, 2].values.
88                                                  reshape(-1, 1)).flatten()
89 testing_features = scaler_features.transform(testing_data.iloc[:, feature_columns].
90                                              values)
91 testing_targets = scaler_targets.transform(testing_data.iloc[:, 2].values.reshape
92                                           (-1, 1)).flatten()
93
94 #Test if it is correct normalized
95 print("Training features range:", training_features.min(), training_features.max())
96 print("Training targets range:", training_targets.min(), training_targets.max())
97
98 print("Testing features range:", testing_features.min(), testing_features.max())
99 print("Testing targets range:", testing_targets.min(), testing_targets.max())

```

```

95
96
97 #Save the Best Model
98 best_model = tf.keras.models.load_model('best_model.h5')
99
100 #Predictions
101 predictions = best_model.predict(testing_features)
102 predictions = scaler_targets.inverse_transform(predictions).flatten()
103 true_values = scaler_targets.inverse_transform(testing_targets.reshape(-1, 1)).
    flatten()
104
105 # Inverse transform the testing features for interpretability if they were scaled
106 testing_features_inverse = scaler_features.inverse_transform(testing_features)
107 # Convert to DataFrame
108 testing_features_df = pd.DataFrame(testing_features_inverse, columns=[f'Feature_{i
    +1}' for i in range(testing_features.shape[1])])
109
110 # Save to CSV
111 testing_features_df.to_csv('testing_features.csv', index=False)
112
113 predictions_df = pd.DataFrame({'True Values': true_values, 'Predictions':
    predictions})
114 predictions_df.to_csv('predictions.csv', index=False)
115
116 tf.keras.models.save_model(best_model, 'central_hole_net_stress_xx', save_format='
    tf')

```

A.7 Annex 7

```

1     import csv
2
3     # Open a CSV file in write mode
4     with open('plate_holes.csv', 'w', newline='') as csvfile:
5         csvwriter = csv.writer(csvfile)
6
7         # Write the header
8         csvwriter.writerow(['Plate Length', 'Plate Width', 'Hole Center X', 'Hole
            Center Y', 'Hole Radius'])
9
10        length = 1.0
11
12        # Loop until length reaches or exceeds 5.0
13        while length <= 5.0:
14            width = 1.0 # Reset width to 1.0 every time length increments
15            while width <= 5.0:
16

```

```

17         x_c = 0.0 # Initialize x-coordinate of the hole
18         while x_c <= length:
19
20             y_c = 0.0 # Initialize y-coordinate of the hole
21             while y_c <= width:
22
23                 r = 0.2 # Starting radius of the hole
24                 while r <= min(length, width) / 2: # Ensure radius doesn't
25                     exceed half of the smallest dimension
26                     # Check conditions for the hole placement and size
27                     if x_c - r >= 0.05 * length and x_c + r <= 0.95 * length
28                         and y_c - r >= 0.05 * width and y_c + r <= 0.95 * width
29                         :
30                         # Write the result to the CSV file
31                         csvwriter.writerow([length, width, x_c, y_c, r])
32                         r += 0.2 # Increment radius by 0.2
33
34             y_c += 0.5 # Increment y_c by 0.5
35
36             x_c += 0.5 # Increment x_c by 0.5
37
38             width += 0.5 # Increment width by 0.5
39
40             length += 0.5 # Increment length by 0.5

```

A.8 Annex 8

```

1  from abaqus import *
2  from abaqusConstants import *
3  from odbAccess import *
4
5  from caeModules import *
6  from driverUtils import executeOnCaeStartup
7
8  import os
9  import csv
10 import numpy as np
11 import math
12 import time
13
14 import mesh
15
16 import sys
17
18 filename = 'plate_holes'
19

```

```

20 mesh_size = 0.05
21
22 folder_path = 'database'
23
24 # Check if the folder exists
25 if not os.path.exists(folder_path):
26     # Create the folder
27     os.makedirs(folder_path)
28
29
30 def get_nodes_on_edge(nodes,dim,x):
31     #n is the array of nodes in the part
32     #dim is if the edge is vertical or horizontal
33     #x is the value of the relevant coordinate of the edge i.e if its y = 2 this
        means dim = 1 x = 2
34     labels = []
35     for n in nodes:
36         if n.coordinates[dim]==x:
37             labels.append(n.label)
38
39     #the returned labels is the tuple that is necessary to create a sequence from
        labels
40     return tuple(labels)
41
42
43 with open(filename+'.csv') as file_obj:
44     os.chdir(folder_path)
45     reader_obj = csv.reader(file_obj) #each line of the file has length [index0],
        height [index1], radius [index2]
46     plate_id=0
47     for row in reader_obj:
48         plate_id = plate_id+1
49         job_name = filename + '_' +str(plate_id)
50         plate_length = float(row[0])
51         plate_height = float(row[1])
52         hole_center_x = float(row[2])
53         hole_center_y = float(row[3])
54         hole_radius = float(row[4])
55
56         model_name = 'Model-'+str(plate_id)
57         job_name = 'job_'+model_name
58
59         os.system('abaqus job='+job_name)
60
61         path_odb = job_name+'.odb'
62
63         while not os.path.exists(path_odb):
64             time.sleep(1)
65

```



```

66         time.sleep(3)
67         odb = openOdb(path_odb)
68
69         os.makedirs(job_name)
70         os.chdir(job_name)
71
72         lastFrame = odb.steps['Step-1'].frames[-1]
73         stress = lastFrame.fieldOutputs['S'].getSubset(position=INTEGRATION_POINT)
74         stress_values=stress.values
75         strain = lastFrame.fieldOutputs['E'].getSubset(position=INTEGRATION_POINT)
76         strain_values=strain.values
77         coord = lastFrame.fieldOutputs['COORD'].getSubset(position =
78             INTEGRATION_POINT)
79         coord_values = coord.values
80         stress_nodal = lastFrame.fieldOutputs['S'].getSubset(position =
81             ELEMENT_NODAL)
82         strain_nodal = lastFrame.fieldOutputs['E'].getSubset(position =
83             ELEMENT_NODAL)
84
85         displacement = lastFrame.fieldOutputs['U'].getSubset(position = NODAL)
86         displacement_values = displacement.values
87
88         n = odb.rootAssembly.instances['PART-1-1'].nodes
89
90         for v,w,x in zip(coord_values, stress_values, strain_values):
91             output = open(job_name+'_GP_data.csv','a')
92             output.write(str(v.data[0])+','+str(v.data[1])+','+str(w.data[0])+','+
93                 str(w.data[1])+','+str(w.data[3])+','+str(x.data[0])+','+str(x.data
94                 [1])+','+str(x.data[3])+','+str(plate_length)+','+str(plate_height)
95                 +','+str(hole_radius)+'\n')
96             output.close()
97
98         displacement_array = []
99         for w,node in zip(displacement_values,n):
100             output = open(job_name+'_displacements_w_coords.csv','a')
101             output.write(str(node.label)+','+str(node.coordinates[0])+','+str(node.
102                 coordinates[1])+','+str(w.nodeLabel)+','+str(w.data[0])+','+str(w.
103                 data[1])+','+str(plate_length)+','+str(plate_height)+','+str(
104                 hole_radius)+'\n')
105             output.close()
106
107         stress_nodes_array = []
108         stress_nodes_id = []
109         strain_nodes_array=[]
110         strain_nodes_id=[]
111         for value1,value2 in zip(stress_nodal.values,strain_nodal.values):

```

```

106         stress_nodes_array.append([value1.data[0],value1.data[1],value1.data
107             [3]])
108         stress_nodes_id.append(value1.nodeLabel)
109         strain_nodes_array.append([value2.data[0],value2.data[1],value2.data
110             [3]])
111         strain_nodes_id.append(value2.nodeLabel)
112
113     largest_node_label = max(stress_nodes_id)
114
115     average_nodal_stresses = []
116     average_nodal_strains = []
117     c=0
118     stress_xx =0
119     stress_yy =0
120     stress_xy =0
121     strain_xx =0
122     strain_yy =0
123     strain_xy =0
124
125     for i in range(1,largest_node_label+1):
126         for j in range(len(stress_nodes_id)):
127             if stress_nodes_id[j]==i:
128                 c = c+1
129                 stress_xx = stress_xx + stress_nodes_array[j][0]
130                 stress_yy = stress_yy + stress_nodes_array[j][1]
131                 stress_xy = stress_xy + stress_nodes_array[j][2]
132                 strain_xx = strain_xx + strain_nodes_array[j][0]
133                 strain_yy = strain_yy + strain_nodes_array[j][1]
134                 strain_xy = strain_xy + strain_nodes_array[j][2]
135
136     stress_xx = stress_xx/c
137     stress_yy = stress_yy/c
138     stress_xy = stress_xy/c
139     strain_xx = strain_xx/c
140     strain_yy = strain_yy/c
141     strain_xy = strain_xy/c
142
143     average_nodal_stresses.append([float(i),stress_xx,stress_yy,stress_xy])
144     average_nodal_strains.append([float(i),strain_xx,strain_yy,strain_xy])
145     c=0
146     stress_xx =0
147     stress_yy =0
148     stress_xy =0
149     strain_xx =0
150     strain_yy =0
151     strain_xy =0
152
153     for i in range(len(average_nodal_stresses)):
154         output = open(job_name+'_nodal_stresses_strains.csv','a')
155         output.write(str(average_nodal_stresses[i][0])+','+str(
156             average_nodal_stresses[i][1])+','+str(average_nodal_stresses[i][2])

```

```

        +','+str(average_nodal_stresses[i][3])+','+str(
        average_nodal_strains[i][0])+','+str(average_nodal_strains[i][1])+
        ','+str(average_nodal_strains[i][2])+','+str(average_nodal_strains[i
        ][3])+','+str(plate_length)+'/'+str(plate_height)+'/'+str(
        hole_radius)+'\n')
152         output.close()
153
154         os.chdir('..')

```

A.9 Annex 9

```

1     import numpy as np
2     import pandas as pd
3     import tensorflow as tf
4     from sklearn.preprocessing import MinMaxScaler
5     import os
6     from sklearn.model_selection import KFold
7     import matplotlib.pyplot as plt
8     import keras
9     from sklearn.model_selection import train_test_split
10    import csv
11
12    from sklearn.metrics import accuracy_score
13
14
15    #Function to save model details
16    def save_model_details(model, filename):
17        with open(filename, 'w') as f:
18            # Print the model summary
19            model.summary(print_fn=lambda x: f.write(x + '\n'))
20            # Access and print the learning rate from the optimizer
21            learning_rate = float(tf.keras.backend.get_value(model.optimizer.
                learning_rate))
22            f.write(f'Learning Rate: {learning_rate}\n')
23
24    # Function to load and process data
25    def load_and_process_data(ids):
26        data = []
27        for idx in ids:
28            dir_path = f'job_Model-{idx+1}'
29            file_path = os.path.join(dir_path, f'job_Model-{idx+1}_GP_data.csv')
30
31            if os.path.exists(dir_path):
32                if os.path.exists(file_path):
33                    temp_data = pd.read_csv(file_path, header=None)
34                    data.append(temp_data)

```

```

35         else:
36             print(f"File for job_Model-{idx+1} does not exist. Skipping...")
37         else:
38             print(f"Directory for job_Model-{idx+1} does not exist. Skipping...")
39
40     if data:
41         return pd.concat(data, ignore_index=True)
42     else:
43         print("No data loaded. Returning empty DataFrame.")
44         return pd.DataFrame()
45
46
47 #Function to save model
48 def create_model(layers, neurons, rate):
49     model = tf.keras.Sequential()
50     model.add(tf.keras.layers.InputLayer(input_shape=(7,)))
51     for _ in range(layers):
52         model.add(tf.keras.layers.Dense(neurons, activation='tanh'))
53     model.add(tf.keras.layers.Dense(5, activation = 'tanh'))
54     model.add(tf.keras.layers.Dense(1, activation = 'linear'))
55     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=rate), loss='mae')
56     return model
57
58 # GPU setup
59 gpus = tf.config.list_physical_devices('GPU')
60 if gpus:
61     try:
62         tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
63         tf.config.experimental.set_memory_growth(gpus[0], True)
64         print(f"Using GPU: {gpus[0].name}")
65     except RuntimeError as e:
66         print(e)
67 else:
68     print("No GPU found. TensorFlow will use CPU.")
69
70
71 #Import Data
72 model_data = pd.read_csv('output.csv', header=None)
73
74 # Split data into training and testing
75 np.random.seed(42)
76 train_ids = np.sort(np.random.choice(model_data.index, size=int(0.7*len(model_data)),
77                                     replace=False))
78 test_ids = np.setdiff1d(model_data.index, train_ids)
79
80 # Load training and testing data
81 training_data = load_and_process_data(train_ids)
82 testing_data = load_and_process_data(test_ids)

```

```
82
83 print("Data Imported")
84
85 feature_columns = [0, 1, 8, 9, 10, 11, 12]
86
87 #Normalize the data
88 scaler_features = MinMaxScaler()
89 scaler_targets = MinMaxScaler()
90
91 training_features = scaler_features.fit_transform(training_data.iloc[:,
    feature_columns].values)
92 training_targets = scaler_targets.fit_transform(training_data.iloc[:, 2].values.
    reshape(-1, 1)).flatten()
93 testing_features = scaler_features.transform(testing_data.iloc[:, feature_columns].
    values)
94 testing_targets = scaler_targets.transform(testing_data.iloc[:, 2].values.reshape
    (-1, 1)).flatten()
95
96 #Test if it is correct normalized
97 print("Training features range:", training_features.min(), training_features.max())
98 print("Training targets range:", training_targets.min(), training_targets.max())
99
100 print("Testing features range:", testing_features.min(), testing_features.max())
101 print("Testing targets range:", testing_targets.min(), testing_targets.max())
102
103
104 # Neural network settings
105 num_hidden_layers_options = [20]
106 neurons_per_layer_options = [32]
107 learning_rates_options = [0.0001]
108 epochs_options = [2000]
109 batch_size_options = [32768]
110
111
112 #K-Fold Cross Validation
113 kf = KFold(n_splits = 3, shuffle = True, random_state = 42)
114 print("Folds are defined")
115
116 best_model = None
117 best_performance = np. inf
118
119 all_mae_histories = []
120 fold_no = 0
121
122 for train_index, test_index in kf.split(training_data):
123     fold_no += 1
124     print(f"Training for fold {fold_no}")
125
126     #Training
```



```
169 #Save the Best Model
170 best_model.save('best_model.h5')
171
172 # Average MSE history for the best model
173 plt.figure(figsize=(10, 6))
174 plt.plot(range(1, len(best_history) + 1), best_history)
175 plt.title('Best Model Validation MAE vs. Epochs')
176 plt.xlabel('Epochs')
177 plt.ylabel('Validation MAE')
178 plt.grid(True)
179 plt.savefig('best_model_mae.png')
180 plt.close()
181
182 #Predictions
183 predictions = best_model.predict(testing_features)
184 predictions = scaler_targets.inverse_transform(predictions).flatten()
185 true_values = scaler_targets.inverse_transform(testing_targets.reshape(-1, 1)).
    flatten()
186
187 predictions_df = pd.DataFrame({'True Values': true_values, 'Predictions':
    predictions})
188 predictions_df.to_csv('predictions.csv', index=False)
189
190 tf.keras.models.save_model(best_model, 'Four_holes_net_stress_xx', save_format='tf'
    )
```