Universidade de Aveiro
2021

Miguel Henrique Fonseca Vieira

Development of artificial intelligence techniques for the calibration of numerical models of materials

Desenvolvimento de técnicas de inteligência artificial para a calibração de modelos numéricos de materiais

Universidade de Aveiro
2021

Miguel Henrique Fonseca Vieira

Development of artificial intelligence techniques for the calibration of numerical models of materials

Desenvolvimento de técnicas de inteligência artificial para a calibração de modelos numéricos de materiais

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de António Gil D'Orey de Andrade Campos, Professor Auxiliar com Agregação, do Departamento de Engenharia Mecânica da Universidade de Aveiro.

Esta dissertação teve o apoio dos projetos CENTRO-01-0145-FEDER-029713 - Programa Operacional Regional do Centro (Centro2020), através do Portugal 2020 e do Fundo Europeu de Desenvolvimento Regional.

o júri / the jury

presidente / president

Prof. Doutor Pedro André Dias Prates
Professor Auxiliar da Universidade de Aveiro

Doutor João Miguel Peixoto Martins
Engenheiro Desenvolvimento da *Siemens Gamesa Renewable Energy Blades, S.A*

Prof. Doutor António Gil D'Orey de Andrade Campos
Professor Auxiliar com Agregação da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

O primeiro e mais sentido agradecimento para os meus pais, Julieta e Joaquim, que me acompanharam e apoiarem durante todo o percurso. O sucesso deste é inteiramente vosso! Um simples "Obrigado", é pouco. Espero um dia poder retribuir tudo o que me deram e ainda continuam a dar, para vós o meu maior apreço.

Ao meu orientador Professor Dr. António Gil d'Orey de Andrade-Campos, pela disponibilidade, paciência e ajuda que sempre mostrou durante a realização deste trabalho.

Aos meus amigos chegados, João, Guedes, Mariana, Pina, Teresa e Tiago que também acompanharam durante o curso. Comigo, levo as incontáveis horas de estudo e trabalho na biblioteca. Obrigado por me terem ajudado e aturado nestes últimos anos.

Ao Rúben Lourenço que me ajudou no desenvolvimento e implementação da parte prática com comentários construtivos e orientação durante a realização deste trabalho.

Por fim, e não menos importante, aos meus tios, avós, primos e demais que fizeram parte deste meu percurso e me ajudaram de uma forma ou de outra a atingir este objectivo.

keywords Artificial Neural Networks, Material model parameter identification, FEA

abstract Nowadays, Finite Element Analysis (FEA) is currently being widely used in the design and development of new parts and their attained results are generally accepted as trustworthy. However, the accuracy of these results highly depends on the ability of the simulation software to reproduce the material's behaviour. So, the calibration of constitutive material models is of extreme importance. Even though the developments in Digital Image Correlation techniques and full-field measurements have proven to be effective for both linear and non-linear models, Finite Element Model Updating (FEMU) and Virtual Fields Method (VFM) are complex and time-consuming. In this work, a new parameter identification methodology is developed, which uses an artificial neural network (ANN) to directly model the inverse problem. The ANN model is trained using data from the FEA direct problem, resulting in an accurate calibration solution. The proposed methodology was validated using two heterogeneous test: a tensile test on a dogbone specimen and the biaxial cruciform test.

palavras-chave Redes neuronais artificiais, Identificação de parâmetros de materiais, FEA

resumo

Actualmente, o uso da Análise de Elementos Finitos (FEA) está a ser utilizado no projecto e desenvolvimento de novas peças e os resultados alcançados são geralmente aceites e confiáveis. No entanto, a precisão desses resultados depende muito da capacidade do software de simulação numérica em reproduzir o real comportamento dos materiais. Portanto, a calibração dos modelos constitutivos de materiais é de extrema importância. Embora os desenvolvimentos em técnicas de correlação digital de imagem e medições de campo completo provem ser eficazes para modelos lineares e não lineares, a actualização do modelo de elementos finitos (FEMU) e o método dos campo virtual (VFM) são complexos e demorados. Neste trabalho, uma nova metodologia de identificação de parâmetros é desenvolvida, que utiliza uma rede neuronal artifical (ANN) para modelar diretamente o problema inverso. O modelo ANN é treinado usando dados do problema direto desenvolvido no FEA, e resulta na calibração precisa da solução. A metodologia proposta é validada usando dois casos de estudo de ensaios heterogéneos: ensaio de tração e um ensaio biaxial cruciforme.

Contents

List of Tables

Intentionally blank page.

List of Figures

Intentionally blank page.

Chapter 1

Introduction

1.1 Framework

Sheet metal forming is one of the most important manufacturing process for obtaining high-performance metal parts for industrial production sectors such as automotive and aerospace. These industries heavily really on numerical simulation tools and the success of these simulations depends on the precision of the input data. So, these industries require extensive knowledge of the material's behaviour, particularly on the choice of appropriate constitutive law and the accurate calibration of its parameters. The inaccuracy of material parameters may lead to flawed results.

Currently, the general state-of-the-art approach to calibrate material model parameters is based on full-field measurements and numerical inverse methods such as the Finite Element Model Updating (FEMU) or the Virtual Fields Method (VFM). Several studies have been made comparing several full-field strategies and although these proven to be effective for both linear and non-linear models, computational time and implementation methodology is still a major disadvantage [3] [4].

The full-field experimental technique that is currently being used and continuously developed is the digital image correlation (DIC) due to new developments in computer technology and higher resolution cameras. This technique allows for the characterization of material parameters far into the range of plastic deformation with fewer experiments required to calibrate a material's model.

Considering the direct problem, which is to obtain the strain, displacement and stresses considering the input parameters such as boundary conditions, geometry, or material parameters, the inverse is to obtain the material parameters and stresses when the input is geometry, boundary conditions, displacements, and strains as shown in the left side of Figure 1.1.

In this context, Machine Learning (ML), a branch of artificial intelligence (AI), can be used to solve the inverse problem in an efficient way. ML can be defined as the automatically improve knowledge of computer algorithms from the provided data and can be used in different areas of science, from medicine to materials science and engineering. Artificial Neural Networks (ANN), a sub-branch of ML, provide accurate and unique solutions based on correlations rather than the very resource intensive optimization procedures. This network replicates the way neurons in brains work.

1.2 Objectives

The present work proposes the calibration of material parameters of elastoplastic constitutive models using artificial intelligence techniques with a focus on artificial neural networks. Its main objectives are: (i) analyse the proposed methodology and compare it with state-of-the-art methodologies. For that purpose, the implementation and validation include: (i) developing a Python Script using Machine Learning that trains with data obtained from the direct problem and does the inverse when receiving experimental data; and (ii) analysis and validation of neural network.

The diagram in 1.1 describes the correlation between the developed methodology and the direct-inverse problem. On the one hand, the direct problem consists in acquiring the stresses (σ), strains (ϵ) and displacements (u) of FEA simulation while knowing the geometry, boundary conditions and the material parameters. On the other hand, the inverse problem consists in discovering the unknown material parameters with the knowledge of geometry, boundary conditions, displacements, and strains.

In short, the main objective of this work is to develop a machine learning model, in this case, a neural network, that is capable to train itself with a synthetic database provided by FEA software of the direct problem and then being able to solve the inverse problem when provided with the displacements and/or strains from a test. The solution of the inverse problem provides the unknown material parameters.

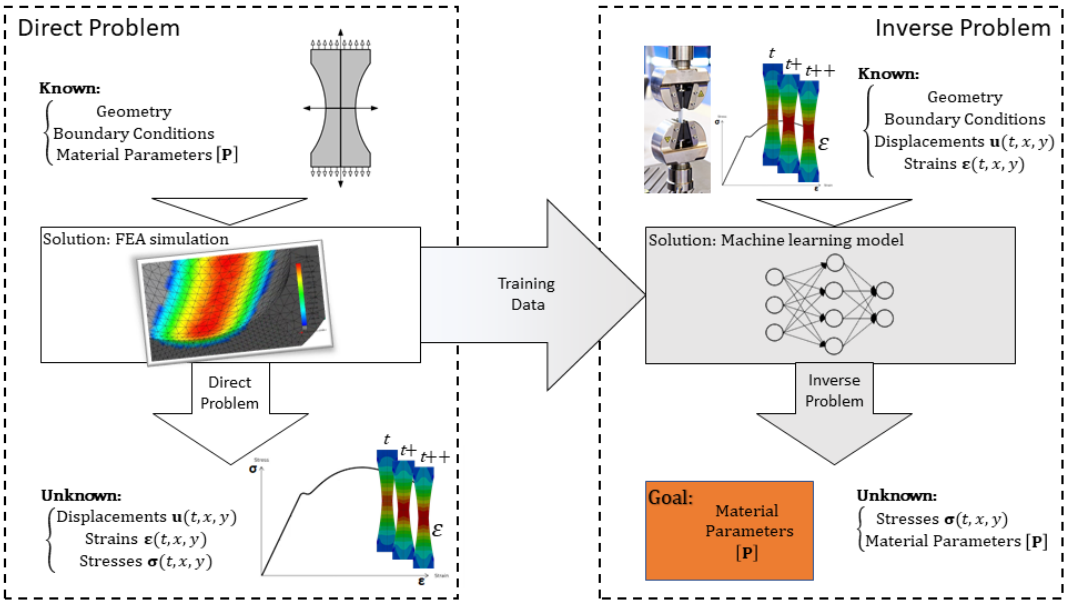


Figure 1.1: Correlation between the direct and inverse problem with the ML model, the training data is gathered from the direct problem and passed to the ML model. Other synthetic data from the inverse problem is then sent to the Model to predict the material parameters.

1.3 Reading Guide

This work is organized as follows: Chapter 2 lays the foundations for this thesis, inverse methodologies and artificial neural networks are boarded in a state-of-the-art review. In Chapter 3 the methodology is discussed followed by a discussion in the development and implementation of the methodology. In Chapter 4, two case studies are presented, a simple tension test and a biaxial test followed by the implementation of the neural network. In Chapter 5 results are presented, discussed and compared with the empirical results. Finally, Chapter 6 concludes this dissertation with some insights about topics and ideas for future research in the area.

Intentionally blank page.

Chapter 2

State-of-the-Art

2.1 Inverse Methodologies for material parameters identification

Nowadays, the mechanical characterization of constitutive laws of materials are of extreme importance on numerical simulations. For the correct characterization of the mechanical behaviour, classic mechanical tests were performed with the objective of material parameters identification. More recently with the development of large and complex non-linear models, and the development of powerful computers and cameras, heterogeneous tests are being used and developed for material parameter identification.

On the one hand, the classical homogeneous testst that have been used in general present a homogeneous strain distribution as stress-strain data, such as tensile test or bulge test. However, the homogeneity of these tests does not resemble the complex stress and strain fields attained in metal forming operations. In fact, for simple constitutive models with a small number of coefficients, these tests seem appropriate. However, for non-linear constitutive models with larger complexity, there are many parameters to be identified resulting in a high number of classical tests needed to be performed. The amount of tests needed results in time cost and expensive material costs [5].

On the other hand, heterogeneous experiments have been developed mainly due to the innovations made in full-field techniques. The need for these experimental tests comes from the necessity to increase the reliability of identified parameters with a reduced number of tests [4]. Therefore, from these heterogeneous tests, several parameters can be identified from a single test if the information is rich enough [6]. Currently, there are several proposed heterogeneous tests and usually, these are modified classical tests by either adding a hole or a notch. A butterfly test was developed in [6] and a FEMU analysis was conducted to identify material parameters comparing them with the reference values. The results showed that the specimen was able to characterize the material behaviour and could effectively promote material parameters identification for complex phenomenological models involving many parameters.

Moreover, DIC techniques have been developed in recent years and these techniques combine image registration and tracking methods for accurate 2D full-field measurements of changes in images. In other words, these methods measure the displacement and strain fields by comparing the pattern of grey levels of the sample surface during deformation with the reference image taken prior to loading [5].

2.2 ANN applied to constitutive modelling

ANN have been studied in mechanical constitutive modelling since the 1990s, when Ghaboussi et al. proposed the constitutive modelling of concrete in [7]. Since then, neural networks have been subjected to huge improvements regarding its application in material modelling such as concrete or sand [7], [8].

Neural networks are computational models that mimic the brain of a human to recognize relationships and associations from a data set and can learn complex non-linear information. The ANN modelling is different from the mathematical modelling of material. In [9], a neural network was incorporated into a FEA program as a replacement to conventional constitutive material model. The work has shown that ANN in comparison with traditional constitutive models that have complex constitutive mathematical models, ANN is not subjected to errors in results since it does not rely on the mathematical description. Also, the network can become more effective and robust as more data becomes available.

The training of an ANN is done by using data gathered from tests performed in the laboratory or computational simulations to learn the material response. There is no need for other idealizations except the physical observations of stress being dependent on strains and its history, i.e. with enough information, the network can generalize the material behaviour to new loading cases [10]. In addition, in [11], an ANN was developed to replace a constitutive model and be incorporated in FEA. The examples provided shown that the neural network can in fact approximate the stress-strain paths.

Currently, neural network experts are debating on how much data is required, and information richness needed for a precise training of a neural network and the influence of noisy data on results. In [12] noise data was introduced to evaluate the performance on neural networks. The authors analysed data from fuzzy structural analysis and the conclusions are that the model can be expanded to consider further material characteristics. In [13], an implicit constitutive model is replaced using a neural network and it's applied as implicit viscoplastic model. The results showed that the neural network as capable of reproducing the original stress-strain curve from the Chaboche's model.

2.3 ANN applied to the calibration of material models

In [14] an inverse material parameter identification is developed, regarding, the application of neural networks to the calibration of models. The ANN model is trained with data from FEA analysis and the validation procedure is done using experimental data. The results showed that the network was capable of identifying the material parameters and friction coefficient.

Moreover, the work done in [15] is regarding the first approach of a simple deep neural network applied to material parameter identification, which is developed and trained with force-displacement curves. The conclusions are that the simple network was able to easily identify parameters and further studies should be made in the area of hyperparameters optimization, more detailed tensile tests and the consideration of more material parameters to be identified simultaneously.

In other works, such as [16] and [17], the author also applies neural networks for the inverse identification of material parameters. In the first study, a bulge test is a base for the creation of the database. The objective is to identify the material parameters using

the force-displacement curve at the central point. The conclusion is that the network easily learns the curves and after the training, the results are almost instantaneously. In the second study, [17], a similar methodology is applied but with the addition of Hill's criterion and the conclusions are the same as the previous study.

In conclusion, machine learning is a wide topic with different applications. In the case of this work, constitutive modelling of materials, there have been several and different approaches to the problem. The works done approach the problem by incorporating ANN in the FEA model. The general results show that there is potential for improving the identification of material parameters, more specific by reducing the labour and time-consuming tests.

This work aims attacking advantage of modern computational power and AI algorithms to discover unknown material parameters. On the one hand, this methodology is easier to implement and faster at obtaining results in comparison with the current VFM and FEMU methods. On the other hand, these algorithms are heavily dependent on the computational power and time-consuming FEA simulations to create a decent database. This work uses the ANN to train and predict material parameters without the need to make more idealizations. The only work done by the FEA is to provide synthetic data for the database. It is expected that the network can discover instantaneously the unknown parameters.

2.4 Machine Learning

Artificial Intelligence (AI) is the simulation process of human intelligence by machines, within this topic, Machine Learning (ML) deals with systems and algorithms that can learn any new data and gain knowledge to achieve better results. The algorithms used in ML can be trained with supervised or unsupervised learning.

On the one hand, unsupervised learning, include training examples, which are not labelled to which class they belong. On the other hand, supervised learning is a machine learning algorithm that use previously-labelled data to learn its features, so they can classify similar but with unlabeled data. Furthermore, within supervised training, regression and classification algorithms can be used. Classification analyses are used for discrete values while regression algorithms are used to predict the continuous values.

Inside the massive topic that machine learning is neural networks were developed to simulate the human nervous system for ML tasks by treating the computational units in a learning model in a manner similar to human neurons and they can derive conclusions from a complex and seemingly unrelated set of information. There are different types of neural networks such as artificial neural networks (ANN) and convolutional neural networks (CNN) and others. [18]

A Artificial Neural Network is a group of multiple neurons, it is also known as a Feed-Forward Neural network since the inputs are processed only in the forward direction. Some of the advantages are the ability to work with incomplete information and to store information in the entire network. However, these networks depend heavily on hardware. Furthermore, ANN uses weights and activation functions for the bulk of its method and these functions introduce nonlinear properties to the network. This helps the network learn any complex relationship between input and output.

Other networks such as Convolution Neural Networks are ideal for tasks such as

image and video recognition. The algorithm takes an input and automatically learns the filters. These filters help in extracting the right and relevant features from the input data without any human supervision.

Despite their differences, both methods use measures of error such as the minimization of a loss function such as the mean square error or mean absolute error, which represents the output error with respect to the desired output system to improve learning.

Chapter 3

Methodology

3.1 General Approach

In the context of this dissertation, an Artificial Neural Network using supervised training with regression analysis is used since the input data will be known *a-priori* and the value of the parameters for the function that best fits an input dataset are continuous. Moreover, according to [19], multi-layer feed-forward network is known to be most suitable for non-linear functions and so far has been the only type of neural network used to describe material constitutive behaviour. So a similar network will be used for the development of this work.

In Figure 3.1 there is a generic representation of a multi-layer feed-forward network with 3 layers with – n-Input, k-Hidden and i-Output. The input layer accepts the input data, the hidden layer processes the inputs, and the output layer produces the result. There are several different types of activation functions and their selection is an important part of the design process of a neural network and is chosen by the designer.

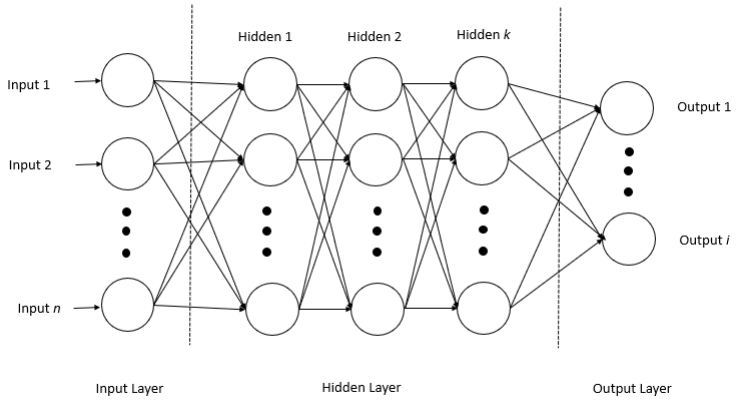


Figure 3.1: Generic Artifical Neural Network

The artificial neuron takes a vector of input features (x_1, x_2, ..., x_n) and each of them is multiplied by a specific weight, (w_1, w_2, ...,). The weighted inputs are summed together, and a constant value called bias (b) is added to them to produce the net input of the neuron, as $z = \sum_{i=1}^{n} x_i w_i + b$.

The result is then passed through an activation function g, such as *sigmoid*, *relu* or *tanh*, to produce the output which is then transmitted to other neurons presented, $a = g(z)$. Although the activation function is chosen by the designer, the w_i and b are adjusted by some learning rule during the training process of the neural network.

In 3.2, there is a visual representation of the general methodology of this work. Has stated, the main objective is to discover the unknown material model parameters from the inverse problem. The ANN model works as the main hub between the direct and inverse problems. It trains itself with data gathered from the FEA software (direct problem) and predicts the material parameters from the strains.

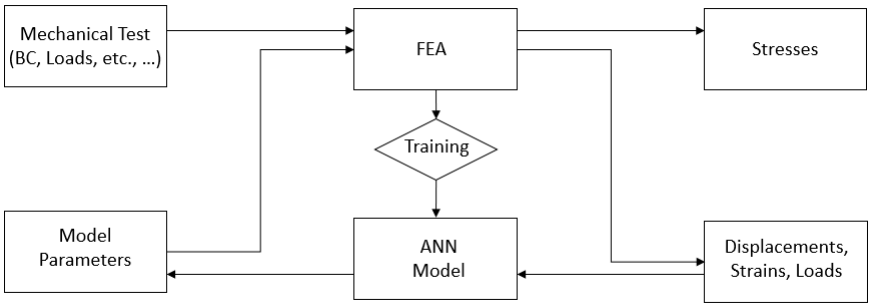


Figure 3.2: General schematic of the methodology.

3.2 Developed Methodology

The development and implementation of the neural network was done in *Python* [20] since this programing language is ideal for general-purpose applications due to being open-source. Furthermore, *Python* allows easy access to libraries that are suited to easier and fast implementation and the main ones used in this work are *Keras* [21] and *Tensorflow* [22]. Both are used to develop and implement reliable machine learning algorithms for scientists and engineers without the need for extensive programming capabilities.

Also, other libraries were used to complement the main packages, such as *Scikit-learn* [23] for pre-processing data and learning-validation curves while *Pandas* [24] and *Numpy* [25] where used for array manipulation and mathematical manipulation respectively.

In short, the proposed identification methodology for the constitutive parameters is based on a multi-layer feed-forward network, shown in Figure 3.3. The synthetic data set is generated by FEA simulations with a variety of material constitutive parameters while maintaining geometry and load conditions. The data is stored in a database where each file represents a different test with different parameters. Then the data is normalized before it is fed to the network for training and validation. Finally, for testing and/or cross-testing some data sets are subjected to a prediction/identification procedure to test the reliability of the network.

The database is created by the FEA model. A test subject is developed with a variety of material parameters while maintaining the geometry, boundary conditions and displacements. This huge data set is then standardized and randomly selected 20% for validation before being fed to the neural network for training. A set of simulations is

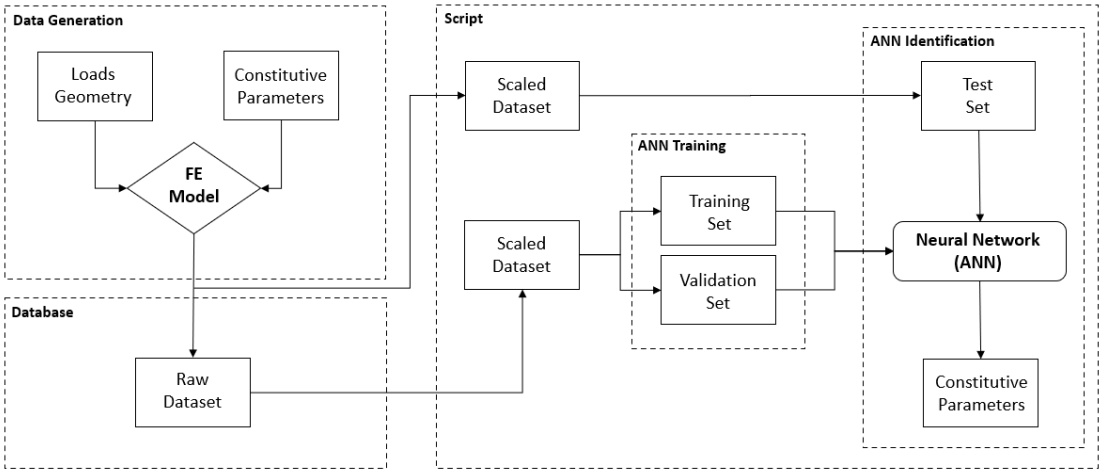


Figure 3.3: Visual representation of the flow of dataset information regarding the metodologhy

also created and scaled for prediction and fed after the completion of the training. After training the predictions are obtained almost instantaneously.

3.3 Applied Methodology

In Figure 3.4 there is a visual representation of the general implementation and development of the neural network. As previous stated, the first step is to develop the automatization for the generation of the database. The associated *Python Script* can be seen in appendixes A.1 and B.1. The database is stored in Comma Separeted Values (CSV) files.

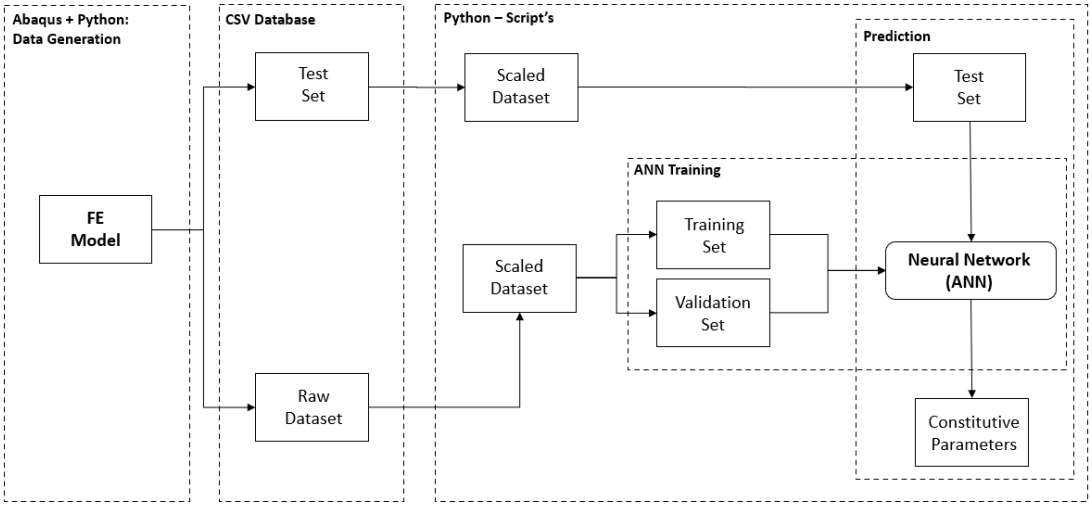


Figure 3.4: Implementation of the methodology

Furthermore, before the creation and search of the hyperparameters for the network,

raw data needs to be normalized since the features of the input data set have large differences in their range even though they have the same unit scale. For this purpose, Maximum Absolute Scaler was used from the *Scikit-learn* library. This algorithm transforms the maximum value to 1 and scales each feature by its maximum absolute value, i.e, $x_{scaled} = \frac{x}{x_{max}}$, where x is the value to scale and x_{max} is the maximum value on the set.

Since a multi-layer feed-forward network will be developed, the *FunctionalAPI* was used from the *Keras* library. The *RandomSearch* algorithm tuner from *Keras Tuner API* class was also used to search the network for the best hyperparameters. This algorithm is easily configured your search space with a define-by-run syntax.

The loss functions used are mean square error and absolute mean error, $MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \overline{Y}_i)^2$ and $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - x_i|)$ respectively, where N is the number of data points, Y_i is the observed value, $\overline{Y}_i$ is the predicted value, y_i is the prediction and x_i is the true value . If the numbers returned by the loss functions are large, then the predictions deviate from the true results meaning the network is not suitable. So, the smaller the error the better the results attained in future predictions.

Gradually, with the help of some optimization function, the loss function learns to reduce the error in prediction. For this purpose, Adam optimizer is employed during the training of the network. This optimization algorithm is based in a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moment and it is known to achieve good results in the field of machine learning.

Chapter 4

Case Studies

In the present chapter, the two case studies are presented. The tests conducted in this chapter using ABAQUS are performed to (i) establish a database for training ANN the algorithm and to (ii) use several tests with known material parameters to validate the neural network.

In the first study, a simple Dogbone test is conducted to identify the parameters from Swift Law's. A test with known parameters is submitted for evaluation and the input geometry, boundary conditions and load conditions are provided from Martins et al. [1]. The results attained are then compared with the results from the same paper.

In the second study, a cruciform test is performed with the same Swift law parameters with the addition of the Hill'48 yield criterion. This model contains 6 material parameters to be identified. This case study is based on the work in [2].

For both studies, the Swift's Law, $\sigma_y = K(\epsilon_0 + \overline{\epsilon}^p)^n$ and $\epsilon_0 = \left(\frac{\sigma_0}{K}\right)^{\frac{1}{n}}$, is used where σ_y is the yield stress, K is the hardening coefficient, n is the hardening exponent, σ_0 the initial yield stress and $\overline{\epsilon}^p$ denotes the equivalent plastic strain. The elastic parameters, Young's modulus (E) and Poisson's ratio (υ) from the material are in Table 4.1.

Table 4.1: Reference elastic material parameters for a generic steel

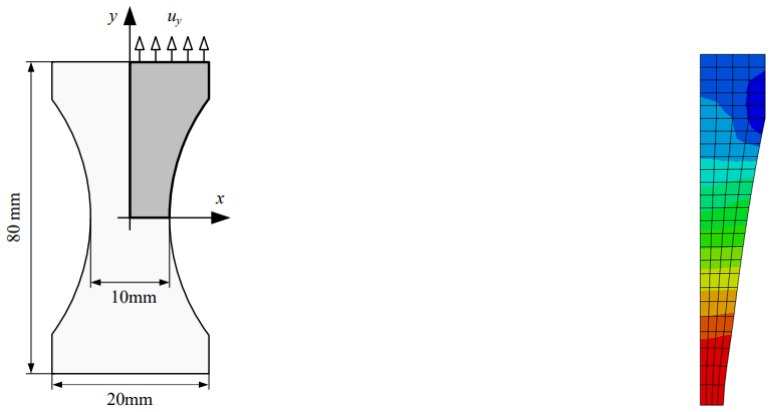
Hooke's Law parameters	**Reference Values**
E[GPa]	210
υ	0.3

All mechanical tests are solved using the finite element software ABAQUS. For standardization, both tests are conducted with twenty-time steps equally spaced. For every entry in the database, the force and the strain in x, y and xy in every nodes is saved. A script file using *Python* was developed to simulate, extract and save the entire database autonomously and efficiently from the simulation software. The code can be viewed in Appendix A.1 and B.1.

4.1 Dogbone Test

The modelling of the dogbone test, represented in Figure 4.1a and the mesh in 4.1b, uses the symmetries of the problem, which means that only one-fourth of the sample is

represented in the FE model as represented in the grey area of the model and the mesh is defined as regular with CPS4R (bilinear shape functions and reduced integration) with a total of 100 elements. Symmetry boundary conditions are applied to the boundaries $x = y = 0$. A boundary condition, with a total displacement of 3 millimetres, is applied along the y direction ($u_y = 3$ mm).



(a) (b)

Figure 4.1: Dogbone tension test: (a) geometry, BC's and dimensions; (b) FE mesh [1]

For the training data set, the range of the three Swift law parameters used for the generation of the database are shown in Table 4.2. The factorial method was adopted to explore the input space of the material parameters, generating 10,879 samples for the training set. The full source code for the automatization of this test can be found in appendix A.1.

Table 4.2: Dogbone Test: input space and steps used for the constitutive parameters for dogbone test

Constitutive parameters	**Input Space**	**Step**
σ_0[MPa]	80-300	10
n	0.1-0.3	0.02
K[MPa]	280-700	10

In addition, to evaluate the performance of the ANN model, a reference set of material parameters is chosen for the Swift's law with the values presented in Table 4.3.

Table 4.3: Reference material parameters for the dogbone test [1]

Swift law parameters	**Reference Values**
σ_0[MPa]	160
n	0.26
K[MPa]	565

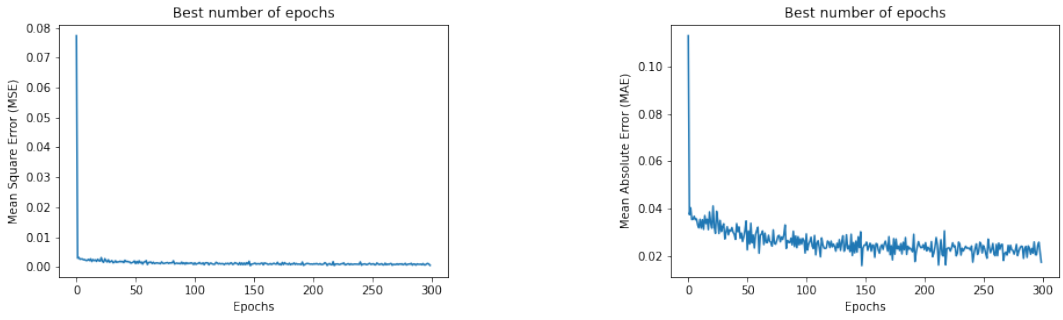
4.1.1 Implementation for the Dogbone test

In brief, the ideal hyperparameters were searched and obtained for the neural network using the *RandomSearch* algorithm from the *Keras* library. Several parameters where searched such as the number of neurons, the activation function, the kernel initializer and the learning rate. The best architecture attained is presented in Table 4.4. Furthermore, the input layer is 8400 since there are four data points (Force, $Strain_x$, $Strain_y$, $Strain_{xy}$) multiplied by the 100 elements and the 21 time steps. The learning rate was also suggested at 1,1775e-5.

Table 4.4: Neural Network architecture for the Dogbone test

Layer	**Neurons**	**Activation**	**Kernel_initializer**
Input	8400	-	-
Dense_1	5000	tanh	Orthogonal
Dense_2	1500	tanh	Variance_scaling
Dense_3	5500	tanh	Truncated_normal
Dense_4	1500	tanh	Truncated_normal
Dense_5	5000	selu	Variance_scaling
Output	3	-	-

The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset and after the architecture being decided, there was a need to decide which is the ideal number of epochs. The aim is to minimize the Mean Square Error Figure 4.2a and to analyse the Mean Absolute Error presented in Figure 4.2b. For that, the training of the network is realized with a high number of epochs and then the minimum is chosen. The figure shows that both errors drop to the 0.001-0.002 and 0.02-0.03 range respectively. The best result was achieved at 143 epochs with a mean squared error of 0.0004 and a mean absolute error of 0.01.



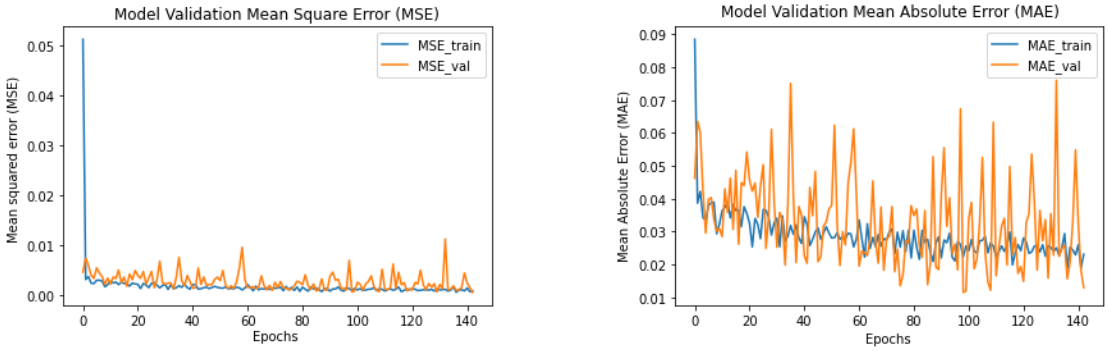
(a) Mean Square Error

(b) Mean Absolute Error

Figure 4.2: The best number of epochs for minimization of MSE and MAE

Figure 4.3a and 4.3b describe train-validation curves from the mean square error and mean absolute errors respectively. Both Figures show validation and a training curve since from the original dataset there is an 80%-20% split in training and validation respectively. These curves help in the diagnosing of the model performance. On the one hand, the MSE plot show an exponential decay in the beginning and a plateau

and an erratic validation curve. This means that, even though the validation curve is inconstant, it still decreases to near-zero and the training curve suggests a correct dimensioning of the network. On the other hand, the MAE chart shows a small decay of the training curve and an larger erratic movement of the validation curve in comparisson with the validation curve of the MSE. This erratic movement shows an unrepresentative validation dataset relative to the training dataset since for correct dimensioning there should be a steady approach between both curves.



(a) Mean Square Error

(b) Mean Absolute Error

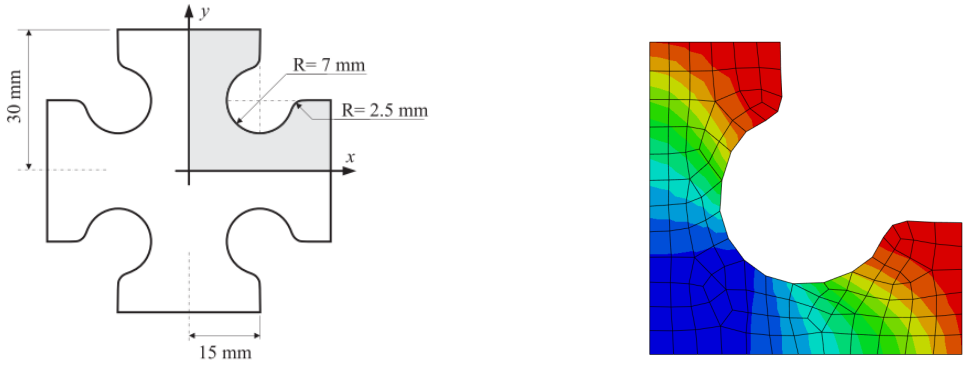
Figure 4.3: Model validation curves for the MSE and MAE - Dogbone

The learning curve determines cross-validated training and test scores for different training set sizes. The cross-validation generator randomly splits the whole dataset 5 times, in training and test sets. Figure 4.4 shows erratic curves in the beginning and then a plateau after the 2500 set samples of the training set. Although the plateau is almost achieved, the analysis of the curves shows that the ANN is lacking some information since near the end there is a negative decline in the training line. This decline indicates that the model is capable of further learning and that the training process was stopped early due to a lack of data. Raising the training set size may solve this problem, tuning the network can also improve the quality of the network.

Figure 4.4: Cross-Validation Curves for the Dogbone architecture

4.2 Cruciform Test

The present study for the cruciform test developed is based in the work done in [2]. In Figure 4.5a there is a visual representation of the full geometry, but for the simulation only one-fourth is used, as represented in the grey area of the model. The mesh is showed in Figure 4.5b, where combination of CPS4 and CPS3R elements, with a total of 114 elements is used. Symmetry boundary conditions are applied to the boundaries $x = y = 0$. A displacement boundary condition, with a total displacement of 2 milimiters, is applied along the x and y direction ($u_x = u_y = 2$ mm).



(a) (b)

Figure 4.5: Cruciform test: (a) geometry, BC's and dimensions; (b) FE mesh [2]

In this case study, for the material model in addition to Swift's law, Hill'48 yield criterion is also considered to describe the behaviour of the material. In addition to the reference material parameters, the Young Modulus and Poisson ration presented previously in Table 4.3 and 4.1 respectively. In Table 4.5, the reference parameters for the plastic anisotropy coefficients (r_α) in the 0^o, 45^o and 90^o directions from the rolling directions are listed.

Table 4.5: Reference plastic anisotropy coefficienss r_α for the biaxial test from [2]

Hill'48 parameters	**Reference Values**
r_0	1.680
r_{45}	1.890
r_{90}	2.253

In this case, these are used as reference for the identification process. These coefficients are the input parameters in order to define potential plasticity in *ABAQUS* software, [26] with R_{11}=R_{23}=R_{31}=1, and can be written as

$$R_{22} = \sqrt{\frac{r_{90}(r_0+1)}{r_0(r_{90}+1)}} \qquad R_{33} = \sqrt{\frac{r_{90}(r_0+1)}{r_0+r_{90}}} \qquad R_{12} = \sqrt{\frac{3(r_0+1)r_{90}}{(2r_{45}+1)(r_0+r_{45})}} \tag{4.1}$$

The range of the six constitutive parameters used for the generation of the database is shown in Table 4.6. Due to the large number of possible combinations, there was

the need to reduce the number of samples of the yield stress, hardening coefficient and hardening exponent in comparison with the previous study. The used method was the Latin Hypercube Sampling and the total number of samples was set at 6000. The source code for this simulation can be found in Appendix B.1

Table 4.6: Input space of constitutive parameters for the biaxial test

Constitutive parameters	**Input Space**
σ_0[MPa]	120-200
n	0.175-0.275
K[MPa]	480-570
r_0	1.0-2.5
r_{45}	1.0-2.5
r_{90}	1.0-2.5

4.2.1 Implementation for the cruciform test

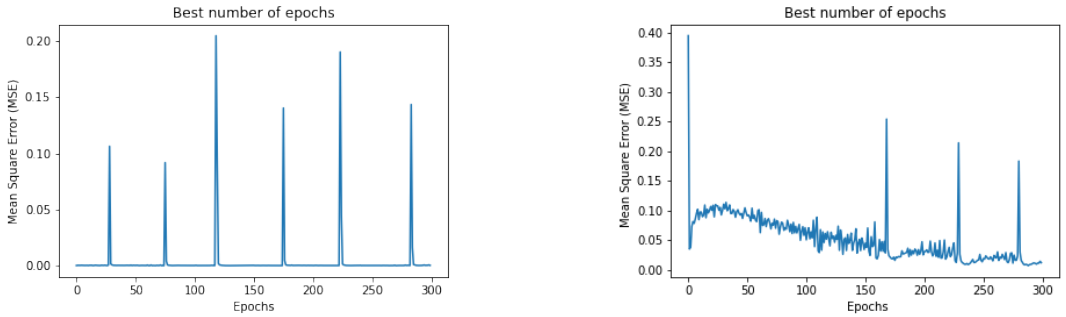
The implementation of the neural network on the cruciform test case study followed the same principles as in the previous one. The ideal hyperparameters were searched and obtained for the network and the ideal architecture is presented in Table 4.7. Furthermore, the input layer is 9576 since there are four data points (Force, $Strain_x$, $Strain_y$, $Strain_{xy}$) multiplied by the 114 elements and the 21 time steps. The suggested learning rate was also attained by the *Hyperband* algorithm at 1,512e-05.

Table 4.7: Ideal Neural Network architecture for the biaxial test

Layer	**Neurons**	**Activation**	**Kernel_initializer**
Input	9576	-	-
Dense_1	7000	tanh	Orthogonal
Dense_2	3000	tanh	Truncated_normal
Dense_3	5000	tanh	Truncated_normal
Dense_4	5000	selu	Variance_scaling
Dense_5	6000	selu	Variance_scaling
Output	6	-	-

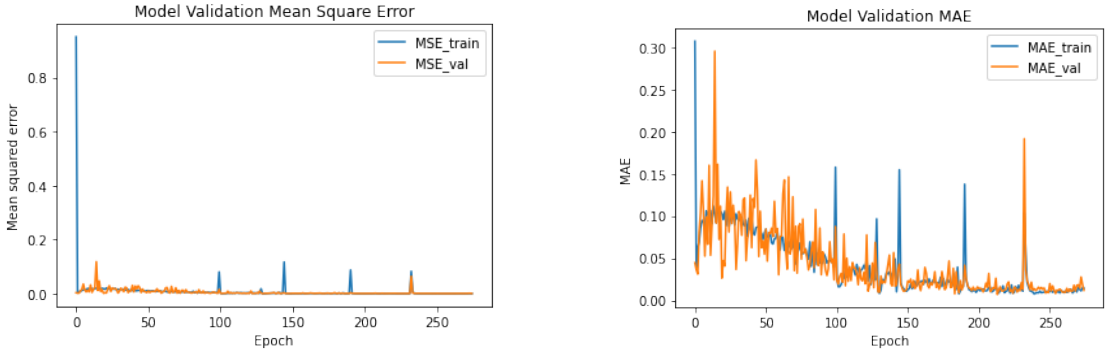
Figure 4.6a and 4.6b presents the Mean Square Error and Mean Absolute Error, respectively. The curves show the ideal number of epochs for training. The graphic shows that both MSE and MAE errors drop to the 0.0001-0.0002 and 0.01-0.003 range respectively. The best result was achieved an 275 epochs with a mean squared error of 0.00083 and a mean absolute error of 0.0072.

In Figure 4.7a and 4.7b there is train-validation curves from the mean square error and mean absolute error respectively. The MSE plot show an exponential decay in the begging and a plateau and an erratic validation curve with a very low error, while the MAE graph shows a small decay of the training curve and an bigger erratic movement of the validation curve. The bigger erratic movement in the MAE curves show as stated before a lack of information on the validation data-set.



(a) (b)

Figure 4.6: The best number of epochs for minimization of MSE and MAE



(a) (b)

Figure 4.7: Model validation curves for the MSE and MAE - Cruciform

Finally for the cross-validation curve in Figure 4.8 never achieves a plateau and the analysis of the curves shows that the network is lacking information. This is a case of underfitting with an unrepresentative dataset. The underfit is due to both curves still are decreasing at the end of the chart. The training data curve is below the validation set which means the data is unrepresentative.

Figure 4.8: Cross-Validation Curves for the biaxial test

Intentionally blank page.

Chapter 5

Results

5.1 Dogbone Test

To evaluate the performance of the neural network, a reference set of material parameters is chosen for Swift's law with the values presented in Table 5.1 and then compared to the results in [1]. In addition, two other simulations were added for further analysis of the capabilities of the network. In total three predictions were conducted and the chosen parameters are presented in Table 5.1.

Table 5.1: Reference parameters used for prediction of the dogbone test

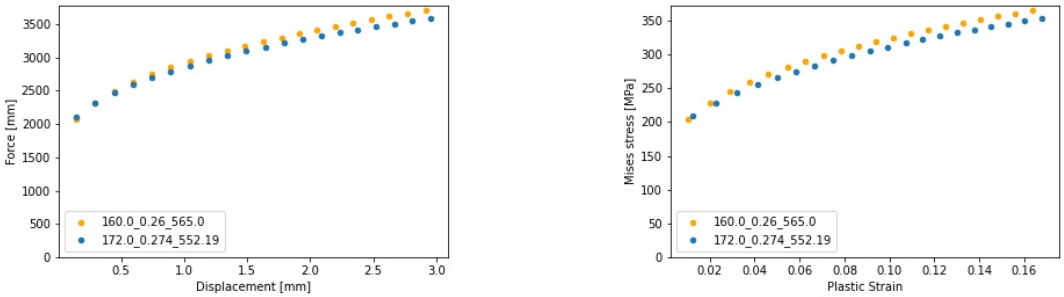
	Reference Parameters		
Swift Parameters	**Simulation 1**	**Simulation 2 (Reference Set)**	**Simulation 3**
$\sigma_0[MPa]$	85.00	160.00	450.00
n	0.11	0.26	0.32
K [MPa]	785.00	565.00	250.00

The prediction achieved using the neural network is listed in Table 5.2. The smallest error was for Simulation 2 (Reference set) in the 2.2-7.5% range, while the largest error was for the Simulation 3 set and it's in the range 17.2-164.1%, which shows a big discrepancy between the control and predicted value.

Table 5.2: Network parameters predictions and corresponding errors for the dogbone test

		$\sigma_0[MPa]$	n	K [MPa]
Simulation 1	**Control**	85.00	0.11	785.00
	Predicted	107.48	0.12	755.41
	Error [%]	26.44	6.36	3.76
Simulation 2 (Reference Set)	**Control**	160.00	0.26	565.00
	Predicted	172.00	0.27	552.19
	Error [%]	7.50	5.38	2.26
Simulation 3	**Control**	450.00	0.32	250.00
	Predicted	527.79	0.49	660.44
	Error [%]	17.28	51.88	164.16

For further analysis, the force-displacement and strain-stress curves are analysed for a reference point with the use of ABAQUS *software*. In fact, where the predicted and the reference stress-strain curves are represented in Figure 5.2a to 5.3b and it can be inferred that even though there is an associated error, the curves diverge from the beginning of the plastic region.



(a) Force - Displacement Curves - Sim 2

(b) Stress - Strain Curves - Sim 2

Figure 5.1: Curves for control and predicted values for the Simulation 2 dataset of the dogbone test

This Simulation is also the reference set used in [1]. In comparison with the study in [1] the error obtain by the neural network is larger than the one from VFM and FEMU methods with fixed increment size, as shown in Table 5.3. Furthermore, the error is in the same range for the FEMU method. The conclusion is that the error for the three methods are very similar, altering only on some parameters.

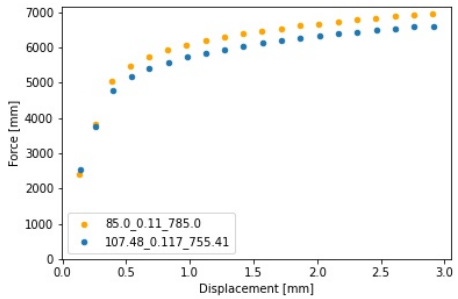
Table 5.3: Comparisson with predictions achieved in [1]

		$\sigma_0[MPa]$	n	K[MPa]
	Control	160.00	0.26	565.00
Sim 2 (Reference)	**Predicted**	172.00	0.27	552.19
	Error [%]	7.50	5.38	2.26
VFM	**Predicted**	160.72	0.26	564.24
	Error [%]	0.46	2.08	0.13
FEMU (fixed increment size)	**Predicted**	160.36	0.26	568.74
	Error [%]	0.23	1.27	0.66

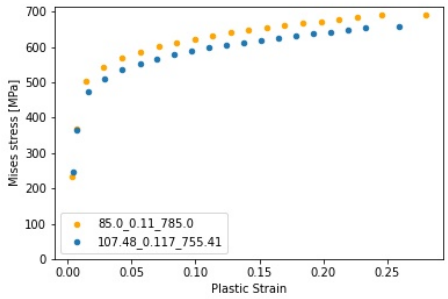
On the one hand in the case of Simulation 1 in Figure 5.2a and 5.2b, the simulation curves achieved have a small gap in the plastic region. This difference is expected since this set of parameters is a mixture between extrapolated data and the corner of the training data space.

On the other hand, in Figure 5.3a and 5.3b relatively to Simulation 3, there is as expected a massive difference between the control and predicted parameters. It's also the only simulation where the control curve is below the predicted parameters. These results can be explained since the control data was outside the training space, meaning the network needed to extrapolate the information to achieve a result.

In conclusion, the ANN network has predicted the material parameters with an error. The error attained was expected according with the different simulations conducted. For

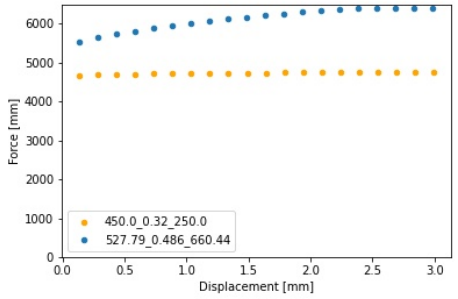


(a) Force-Displacement Curves - Sim 1

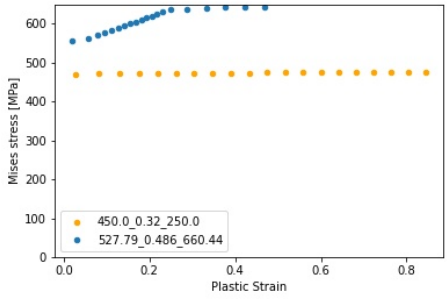


(b) Stress-Strain Curves - Sim 1

Figure 5.2: Curves for control and predicted values for the Simulation 1 dataset of the dogbone test



(a) Force-Displacement Curves - Sim 3



(b) Stress-Strain Curves - Sim 3

Figure 5.3: Curves for control and predicted values for the Simulation 3 dataset of the dogbone test

the Simulation 2, since the training data was developed with this value in the middle it achieved the smaller error. While the Simulation 1 demanded a extrapolation of a parameter and Simulation 3 was completely outside the training space to test the capabilities of the network.

To improve the reliability of the network, further and improved data should be fed to improve the quality of the predictions. Fortunately, since this test isn't computational expensive, the FEA *software* only takes around one minute to perform and save a test so this can be easily expanded as desired. On the other hand, the learning of the data takes another four hours for the ten thousand samples. If the database increases so does the training. After that, the predictions are almost instantaneously. Furthermore, the network architecture must be tested and improved with different combinations of *Hyperparameters* in order to achieve a smaller error.

5.1.1 Sensitivity analysis

Training with predictions

In order to test the reliability of the network, a simple sensitivity analysis was developed. In this case, to the training samples used, the predicted samples were also added to

the training pool, followed by the prediction of those same predicted simulations. The objective is to analyse if the network can learn better since the prediction simulations are in the training samples. In Table 5.4 there are the new predicted values and its respective associated errors.

Table 5.4: Training with predictions and corresponding errors

		σ_0*[MPa]*	n	K[MPa]
Simulation 1	**Control**	85.00	0.11	785.00
	Predicted	103.75	0.10	748.70
	Error [%]	22.06	5.45	4.62
Simulation 2 (Reference Set)	**Control**	160.00	0.26	565.00
	Predicted	165.56	0.27	565.37
	Error [%]	3.48	3.07	0.06
Simulation 3	**Control**	450.00	0.32	250.00
	Predicted	521.78	0.25	536.48
	Error [%]	15.95	23.13	114.59

The new results support the conclusions presented previous, that the accuracy and robustness of the network largely depend on the quality and quantity of the data. In this case, it's obvious the large reduction with the associated error across almost all parameters in the simulations. The most notorious was in the reference set, Simulation 2, where the error was changed to half of the previous simulations on the initial yield stress and hardening coefficient while it's almost zero in the hardening exponent.

Noise Influence

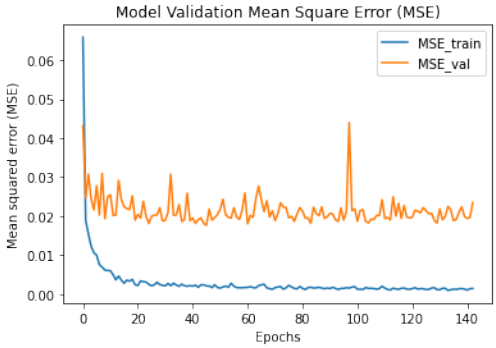
Further analysis was conducted by adding noise to the data set. To every training sample, a random normal distribution with 0 mean and a standard deviation of 1 was added. Table 5.5 gathers the predictions of the network after the training with the polluted data. The predictions attained suffered a larger error compared with the clean data.

Table 5.5: Network parameters predictions and corresponding errors with noise

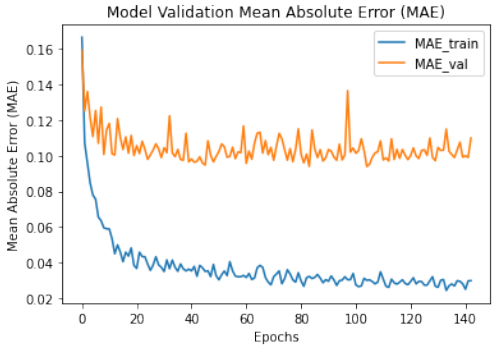
		σ_0*[MPa]*	n	K[MPa]
Simulation 1	**Control**	85.00	0.11	785.00
	Predicted	87.71	0.08	744.74
	Error [%]	3.18	29.09	5.13
Simulation 2 (Reference Set)	**Control**	160.00	0.26	565.00
	Predicted	144.04	0.22	525.98
	Error [%]	9.98	16.15	6.90
Simulation 3	**Control**	450.00	0.32	250.00
	Predicted	365.09	0.25	548.83
	Error [%]	18.87	23.13	119.53

Even though the database was polluted, the error for the reference set was smaller than expected, which shows that even for polluted data the network can learn from it. In

addition, Figure 5.4a and 5.4b shows the history plot of the MSE and MAE respectively. The train and validation curves show some improvement but a large gap between both of them remains. This is a case of an unrepresentative training dataset



(a) Force-Displacement Curves - Noise



(b) Stress-Strain Curves - Noise

Figure 5.4: Curves for control and predicted values for the Simulation 3 dataset of the dogbone test with noise

5.2 Cruciform Test

In order to evaluate the performance of the neural network, the same methodology as Section 5.1 is used. Three reference sets of material parameters are chosen and listed in Table 5.6. The predictions and corresponding errors are depicted in Table 5.7.

Table 5.6: Reference parameters used for prediction of the cruciform test

	Reference Parameters		
Constitutive Parameters	**Simulation 1**	**Simulation 2 (Reference Set)**	**Simulation 3**
$\sigma_0[MPa]$	85.00	160.00	450.00
n	0.11	0.26	0.32
K[MPa]	785.00	565.00	250.00
r_0	1.01	1.68	2.52
r_{45}	1.24	1.89	2.67
r_{90}	1.73	2.253	2.95

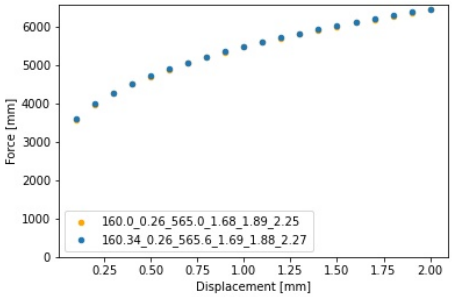
The smallest error was in Simulation 2 (Reference set), while the largest error was for the Simulation 3 set. The error analysis shows an expected result since the lowest error is in the centre of the training samples, while the largrest error is outside the training space. Furthermore, the error in Simulation 1 and 2, is very low which contrasts with the first conclusions discussed in Section 4.2.1 since such low error was not expected.

The curves associated with the force-displacement and strain-stress, in Figures 5.5a and 5.5b, respectively, are identical since the relative error was almost non-existent.

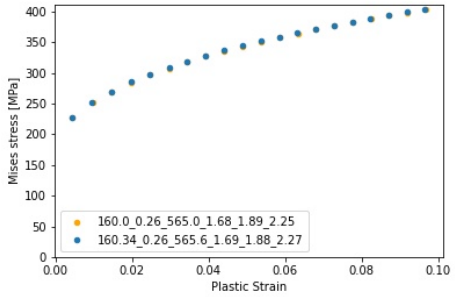
In comparisson with [2] the results show a minimal difference between both errors. This means, that the ANN can predict parameters with the same precision as the VFM method. In Table 5.8 is gathered the predictions achieved in this work and in [2]

Table 5.7: Predictions and corresponding errors for cruciforme test

		σ_0[MPa]	n	K[MPa]	r_0	r_{45}	r_{90}
Sim 1	**Control**	120.00	0.18	530.00	1.01	1.24	1.73
	Predicted	120.98	0.18	536.69	1.06	1.25	1.71
	Error [%]	0.82	1.67	1.26	4.95	0.81	1.16
Sim 2 (Reference)	**Control**	160.00	0.26	565.00	1.68	1.89	2.253
	Predicted	160.34	0.26	565.60	1.69	1.88	2.27
	Error [%]	0.21	1.15	0.11	0.60	0.53	0.75
Sim 3	**Control**	220.00	0.30	620.00	2.52	2.67	2.95
	Predicted	202.61	0.24	558.54	2.36	2.30	2.74
	Error [%]	7.90	21.67	9.91	6.35	13.86	7.12



(a) Force-Displacement Curves- Sim 2



(b) Stress-Strain Curves - Sim 2

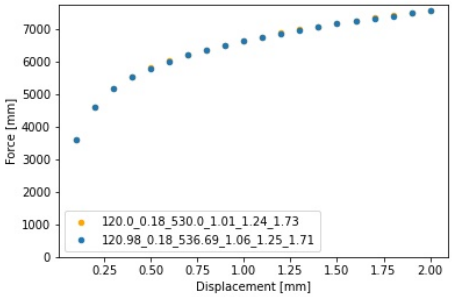
Figure 5.5: Curves for control and predicted values for the Simulation 2 dataset of the cruciform test

Table 5.8: Comparisson with predictions achieved in [2]

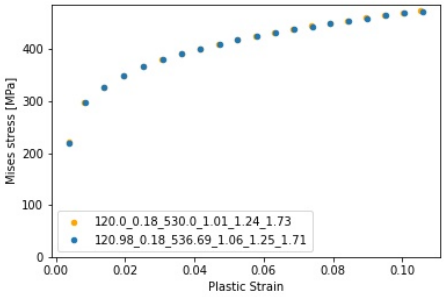
		σ_0[MPa]	n	K[MPa]	r_0	r_{45}	r_{90}
	Control	160.00	0.26	565.00	1.68	1.89	2.25
Sim 2 (Reference)	**Predicted**	160.34	0.26	565.60	1.69	1.88	2.27
	Error [%]	0.21	1.15	0.11	0.60	0.53	0.75
VFM	**Predicted**	159.76	0.26	566.16	1.67	1.91	2.24
	Error [%]	0.15	0.44	0.25	0.16	1.11	0.26

In the case of Simulation 1 in Figure 5.6a and 5.6b, the simulation curves achieved are overlapping in the elastic region while in the plastic region there is a small gap between them on the strain-stress curve. This difference can be explained due to this set of parameters being in the corner of the training data space. Even though this prediction is on the corner of the data-training samples, the network was able to predict it almost perfect.

For the extrapolation case, in Figure 5.7a and 5.3b relatively to Simulation 3, there is a difference between the control and predicted parameters. It's also the only simulation where the control curve is below the predicted parameters. These can be explained since the control data was outside the training space, meaning the network needed to extrapolate the information to predict a result.

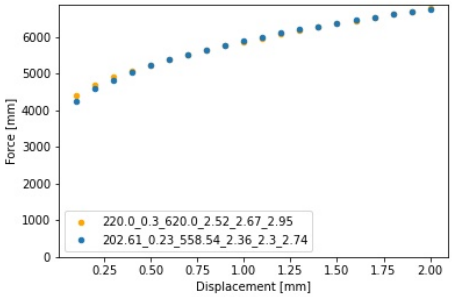


(a) Force-Displacement Curves - Sim 1

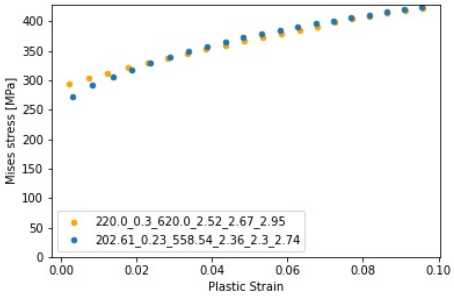


(b) Stress-Strain Curves - Sim 1

Figure 5.6: Curves for control and predicted values for the Simulation 1 dataset of the cruciform test



(a) Force-Displacement Curves - Sim 3



(b) Strain-Stree Curves - Sim 3

Figure 5.7: Curves for control and predicted values for the Simulation 3 dataset of the cruciform test with noise

In conclusion, the network has predicted the material parameters with a very small to non-existent relative error for Simulation 2. The error attained can be explained according to the position of the prediction in comparison with the database. For the Simulation 2, since the training data was developed with this value in the middle it achieved the smaller error.

5.2.1 Sensitivity analysis

Training with predictions

To test the reliability of the network, the predictions sets were added to the training samples just like in the previous section. In Table 5.9 there is the new predicted values and its respective associated erros. In this case study, the most notourious result is the zero error in r_0, all the other parameters have generaly seen their associated error reduce in comparison with the previous prediction except the K and r_{90} parameters.

Table 5.9: Preditions with predictions sets added to the training pool

		σ[MPa]	n	K[MPa]	r_0	r_{45}	r_{90}
Sim 1	**Control**	120.00	0.18	530.00	1.01	1.24	1.73
	Predicted	120.34	0.17	528.00	0.99	1.20	1.79
	Error [%]	0.28	4.44	0.38	1.98	3.23	3.47
Sim 2 (Reference)	**Control**	160.00	0.26	565.00	1.68	1.89	2.25
	Predicted	160.14	0.26	547.73	1.68	1.93	2.39
	Error [%]	0.09	0.38	3.06	0.00	2.12	6.08
Sim 3	**Control**	220.00	0.30	620.00	2.52	2.67	2.95
	Predicted	206.20	0.24	513.11	2.30	2.29	2.82
	Error [%]	6.27	19.00	17.24	8.73	14.23	4.41

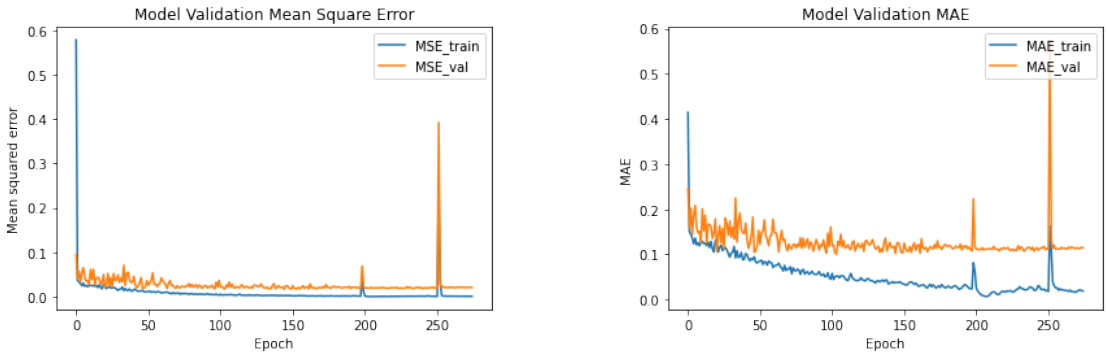
Noise Influence

The same random normal distribution with 0 mean and a standard deviation of 1 was added to the training dataset. Table 5.10 gathers the predictions of the network after the training with the polluted data. The predictions attained suffered a larger error compared with the use of clean data. Even though the network was polluted, the error for the reference set was smaller than expected, which shows that even for polluted data the network can learn from it, but it's not able to predict with reliability.

Table 5.10: Predictions with cruciform noise dataset

		σ_0[MPa]	n	K[MPa]	r_0	r_{45}	r_{90}
Sim 1	**Control**	120.00	0.18	530.00	1.01	1.24	1.73
	Predicted	146.82	0.197	568.02	1.66	1.72	1.98
	Error	22.35	9.44	7.17	64.36	38.71	14.45
Sim 2 (Reference)	**Control**	160.00	0.26	565.00	1.68	1.89	2.25
	Predicted	152.79	0.23	558.82	1.66	1.72	1.98
	Error	4.51	10.00	1.09	8.93	8.47	18.77
Sim 3	**Control**	220.00	0.30	620.00	2.52	2.67	2.95
	Predicted	166.61	0.23	554.62	1.77	1.73	1.93
	Error	24.27	24.27	10.55	29.76	35.21	34.58

In addition, Figure 5.8a and 5.8b shows the history plot of the MSE and MAE respectively, the train and validation curves show some improvement followed by a separation between both of the curves. This is a case of an unrepresentative training dataset in conjunction with an unrepresentative validation set.



(a) Force-Displacement Curves

(b) Strain-Stree Curves

Figure 5.8: Curves for control and predicted values for the Simulation 3 dataset of the cruciform test with noise

Intentionally blank page.

Chapter 6

Final Considerations

6.1 Conclusions and Future Works

The main goal of this dissertation was to develop an AI methodology applied to the calibration of numerical models of materials. A simple and easy to implement feed-forward network was researched and deployed with the help of thousands of FEA numerical simulations from two different tests. The neural network developed in *Python* was able to achieve its objectives and the results show that the network can predict material parameters, even though with some errors.

In the process to achieve the desired solution, several networks and algorithms were tested and researched to achieve the best results possible. The neural network algorithm can be applicable to other test subjects with little to none modification regarding the source code.

There were several challenges regarding the implementation of the network. The main one was the use of the Hyperband algorithm presented in the Keras package. This algorithm may not be ideal since it researches a space defined by the user. The algorithm aims to minimize the loss function and even though they could find that in that specific space, nothing guarantees that the best result attained is the global minimum of the function. Furthermore, this type of algorithm consumes a lot of time and computer resources. In this specific case, the research for the ideal architecture takes around forty eight hours, followed by another six to eight hours just to find the ideal number of epochs and training before being able to predict values.

On the one hand, the neural network model for the dogbone test was able to predict the test sets although with some associated relative error. In short, a simple dogbone test combined with a simple neural network was able to predict the results with a straightforward implementation without the need for complex programming skills.

On the other hand, the cruciform test was also able to predict the set of material parameters. In addition to the simple Swift's Law parameters presented in the previous study, to this case was added Hill's 48 yield criterion to make the network more complex and demanding. The results have shown a much better prediction in comparison with the dogbone test. It is necessary to highlight that the cruciform test is more rich, being that one of the main reasons for the better results.

Furthermore, both sensitive tests conducted during the case studies verify the fact that the reliability of the network is highly dependent on the quantity and quality of the data. The two sensitive tests were performed, first with the addition of the prediction

to the training pool and second with the addition of noise to the training samples. For the first case, the results were in fact better, since the network was subjected to those tests previously although the results weren't perfect. In the second case, the addition of noise has shown that the network is still easily able to learn and predict but with a loss in precision.

Even though the main goal of this dissertation, along with the necessary milestones, were effectively reached, mainly the creation of an AI method and its testing, some improvement can yet be executed or improved.

This dissertation also lays the foundations for several other works regarding the application of artificial intelligence methods in the calibration of numerical models of materials or any topic regarding mechanical engineering. In fact, in future works, it may be interesting to expand this research to deep learning since it automates much of the feature extraction, thus eliminating some of the human intervention. Also, focusing the developing and testing different types and combinations of hyper-parameters of the network. Furthermore, it may be interesting in conducting a live test and have the network trying to predict the material parameters. On the broader side, it may be interesting to experiment with other fields of artificial intelligence to achieve other insights and conclusions regarding the application of modern computing in traditional fields of engineering.

Bibliography

[1] J. Martins, S. Thuillier, and A. Andrade-Campos, "Identification of material parameters for plasticity models: A comparative study on the finite element model updating and the virtual fields method," in *AIP Conference Proceedings*, vol. 1960, no. 1. AIP Publishing LLC, 2018, p. 110007.

[2] J. Martins, A. Andrade-Campos, and S. Thuillier, "Calibration of anisotropic plasticity models using a biaxial test and the virtual fields method," *International Journal of Solids and Structures*, vol. 172, pp. 21–37, 2019.

[3] J. M. P. Martins, A. Andrade-Campos, and S. Thuillier, "Comparison of inverse identification strategies for constitutive mechanical models using full-field measurements," *Int J Mech*, vol. **45**, pp. 340–345, 1992.

[4] S. Avril, M. Bonnet, A.-S. Bretelle, M. Grédiac, F. Hild, P. Ienny, F. Latourte, D. Lemosse, S. Pagano, E. Pagnacco, and F. Pierron, "Overview of identification methods of mechanical parameters based on full-field measurements," *Int J Mech*, vol. **48**, pp. 381–402, 2008.

[5] N. Souto, "*Computational design of a mechanical test for material characterization by inverse analysis*," PhD Thesis, PhD Thesis in Mechanical Engineering, Department of Mechanical Engineering, Aveiro University, Aveiro, 2015.

[6] N. Souto, A. Andrade-Campos, and S. Thuillier, "Mechanical design of a heterogeneous test for material parameters identification," *International Journal of Material Forming*, vol. 10, no. 3, pp. 353–367, 2017.

[7] J. Ghaboussi, J. Garrett Jr, and X. Wu, "Knowledge-based modeling of material behavior with neural networks," *Journal of engineering mechanics*, vol. 117, no. 1, pp. 132–153, 1991.

[8] J. Ghaboussi and D. Sidarta, "New nested adaptive neural networks (nann) for constitutive modeling," *Computers and Geotechnics*, vol. 22, no. 1, pp. 29–52, 1998.

[9] A. Javadi, T. Tan, and M. Zhang, "Neural network for constitutive modelling in finite element analysis," *Computer Assisted Mechanics and Engineering Sciences*, vol. 10, no. 4, pp. 523–530, 2003.

[10] S. Jung and J. Ghaboussi, "Neural network constitutive model for rate-dependent materials," *Computers & Structures*, vol. 84, no. 15-16, pp. 955–963, 2006.

[11] M. Lefik and B. A. Schrefler, "Artificial neural network as an incremental non-linear constitutive model for a finite element code," *Computer methods in applied mechanics and engineering*, vol. 192, no. 28-30, pp. 3265–3283, 2003.

[12] S. Freitag, W. Graf, and M. Kaliske, "A material description based on recurrent neural networks for fuzzy data and its application within the finite element method," *Computers & Structures*, vol. 124, pp. 29–37, 2013.

[13] J.-S. Lee, H.-J. Lee, and T. Furukawa, "Formulation of the neural network for implicit constitutive model (i): Application to implicit vioscoplastic model," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 9, no. 3, pp. 191–197, 2009.

[14] Z. Ktari, C. Leitão, P. A. Prates, and A. Khalfallah, "Mechanical design of ring tensile specimen via surrogate modelling for inverse material parameter identification," *Mechanics of Materials*, vol. 153, p. 103673, 2021.

[15] D. Koch and A. Haufe, "First steps towards machine-learning supported material parameter identification," 2019.

[16] A. Chamekh, H. B. H. Salah, and R. Hambli, "Inverse technique identification of material parameters using finite element and neural network computation," *The International Journal of Advanced Manufacturing Technology*, vol. 44, no. 1-2, p. 173, 2009.

[17] A. Chamekh, H. Belhadjsalah, R. Hambli, and A. Gahbiche, "Inverse identification using the bulge test and artificial neural networks," *Journal of Materials processing technology*, vol. 177, no. 1-3, pp. 307–310, 2006.

[18] Analyticsvidhya. (2018) Cnn vs. rnn vs. ann – analyzing 3 types of neural networks in deep learning. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

[19] Y. Hashash, S. Jung, and J. Ghaboussi, "Numerical implementation of a neural network based material model in finite element analysis," *International Journal for numerical methods in engineering*, vol. 59, no. 7, pp. 989–1005, 2004.

[20] Python. (1991) Python docs. [Online]. Available: https://www.python.org/doc/

[21] Keras. (20015) Keras api reference. [Online]. Available: https://keras.io/api/

[22] TensorFlow. (2015) Tensorflow docs. [Online]. Available: https://www.tensorflow.org/

[23] scikit learn. (2008) Machine learning in python. [Online]. Available: https://scikit-learn.org/stable/index.html

[24] Pandas. (2008) Pandas docs. [Online]. Available: https://pandas.pydata.org/docs/

[25] Numpy. (2006) Numpy docs. [Online]. Available: https://numpy.org/doc/stable/

[26] A. Inc. (2008) Anisotropic yield/creep. [Online]. Available: https://abaqus-docs.mit.edu/2017/English/SIMACAEMATRefMap/simamat-c-anisoyield.htm

Appendix A

Source Code Dogbone

A.1 Abaqus - Dogbone Test

```
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from odbAccess import *
from abaqus import *
from abaqusConstants import *
import visualization

#--------------Definition of some constants-----------------------
K = 280
sigma_zero = 90
N = 0.26

Young_modulos = 210000
poisson = 0.3

#--------------Definition of Swift Law-----------------------------
def swift(valor_k,valor_n, sigma_zero, def_plast):

    def_zero = (sigma_zero / valor_k) ** (1./valor_n)

    sigma = valor_k * (def_zero + def_plast) ** valor_n

    return sigma


#--------------Definition of the part-----------------------------
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=100.0)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(0.0, 50.0))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=False,
    entity=mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 50.0), point2=(10.0, 50.0))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(addUndoState=False,
    entity=mdb.models['Model-1'].sketches['__profile__'].geometry[3])
```

```
43 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(addUndoState=False,
       entity1=mdb.models['Model-1'].sketches['__profile__'].geometry[2],
       entity2=mdb.models['Model-1'].sketches['__profile__'].geometry[3])
44 mdb.models['Model-1'].sketches['__profile__'].Line(point1=(10.0, 50.0), point2=(10.0, 40.0))
45 mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=False,
       entity=mdb.models['Model-1'].sketches['__profile__'].geometry[4])
46 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(addUndoState=False,
       entity1=mdb.models['Model-1'].sketches['__profile__'].geometry[3],
       entity2=mdb.models['Model-1'].sketches['__profile__'].geometry[4])
47 mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(5.0, 0.0))
48 mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(addUndoState=False,
       entity=mdb.models['Model-1'].sketches['__profile__'].geometry[5])
49 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(addUndoState=False, entity1=
50 mdb.models['Model-1'].sketches['__profile__'].geometry[2], entity2=
51 mdb.models['Model-1'].sketches['__profile__'].geometry[5])
52 mdb.models['Model-1'].sketches['__profile__'].Arc3Points(point1=(5.0, 0.0), point2=(10.0, 40.0),
       point3=(6.5, 18.5))
53 mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR, name='Part-1', type=DEFORMABLE_BODY)
54 mdb.models['Model-1'].parts['Part-1'].BaseShell(sketch=
55 mdb.models['Model-1'].sketches['__profile__'])
56 del mdb.models['Model-1'].sketches['__profile__']
57
58 #--------------Definition of the mesh-----------------------------
59 mdb.models['Model-1'].parts['Part-1'].seedPart(deviationFactor=0.1,
60     minSizeFactor=0.1, size=2)
61 mdb.models['Model-1'].parts['Part-1'].generateMesh()
62
63 #--------------Definition of the assembly-----------------------------
64 mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
65 mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='Part-1-1',
66     part=mdb.models['Model-1'].parts['Part-1'])
67
68 #--------------Definition of the step-----------------------------
69 mdb.models['Model-1'].StaticStep(name='Step-1', previous='Initial')
70 mdb.models['Model-1'].steps['Step-1'].setValues(initialInc=0.025, maxNumInc=200)
71 mdb.models['Model-1'].steps['Step-1'].setValues(minInc=1e-20)
72
73 #----------Definition of the boundary conditions------------------
74 mdb.models['Model-1'].rootAssembly.Set(edges=
75     mdb.models['Model-1'].rootAssembly.instances['Part-1-1'].edges.getSequenceFromMask(
76     ('[#10 ]', ), ), name='Set-1')
77 mdb.models['Model-1'].XsymmBC(createStepName='Step-1', localCsys=None, name=
78     'Simetria XX', region=mdb.models['Model-1'].rootAssembly.sets['Set-1'])
79 mdb.models['Model-1'].rootAssembly.Set(edges=
80     mdb.models['Model-1'].rootAssembly.instances['Part-1-1'].edges.getSequenceFromMask(
81     ('[#1 ]', ), ), name='Set-2')
82 mdb.models['Model-1'].YsymmBC(createStepName='Step-1', localCsys=None, name=
83     'Simetria YY', region=mdb.models['Model-1'].rootAssembly.sets['Set-2'])
84 mdb.models['Model-1'].rootAssembly.Set(edges=
85     mdb.models['Model-1'].rootAssembly.instances['Part-1-1'].edges.getSequenceFromMask(
86     ('[#8 ]', ), ), name='Set-3')
87 mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Step-1',
88     distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
89     'Deslocamento', region=mdb.models['Model-1'].rootAssembly.sets['Set-3'],
90     u1=UNSET, u2=3.0, ur3=UNSET)
91
92 #--------------Definition of the Section-----------------------------
93 mdb.models['Model-1'].HomogeneousSolidSection(material='Steel', name=
94     'Section-1', thickness=None)
95 mdb.models['Model-1'].parts['Part-1'].Set(faces=
96     mdb.models['Model-1'].parts['Part-1'].faces.getSequenceFromMask(('[#1 ]',
97     ), ), name='Set-1')
98 mdb.models['Model-1'].parts['Part-1'].SectionAssignment(offset=0.0,
99     offsetField='', offsetType=MIDDLE_SURFACE, region=
100    mdb.models['Model-1'].parts['Part-1'].sets['Set-1'], sectionName=
101    'Section-1', thicknessAssignment=FROM_SECTION)
102
```

```
#--------------Definition of the Field Output-----------------------------
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(variables=(
    'S', 'PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF'))
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(variables=(
    'S', 'PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF', 'TF'))
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(numIntervals=20)

#--------------Definition of the material-----------------------------
mdb.models['Model-1'].Material(name='Steel')
mdb.models['Model-1'].materials['Steel'].Elastic(table=((Young_modulos, poisson), ))
mdb.models['Model-1'].materials['Steel'].Plastic(table=((Young_modulos, 0.0),
                                    (swift(K,N ,sigma_zero,0.01), 0.01),
                                    (swift(K,N, sigma_zero,0.02), 0.02),
                                    (swift(K,N, sigma_zero,0.03), 0.03),
                                    (swift(K,N, sigma_zero,0.04), 0.04),
                                    (swift(K,N, sigma_zero,0.05), 0.05),
                                    (swift(K,N, sigma_zero,0.06), 0.06),
                                    (swift(K,N, sigma_zero,0.07), 0.07),
                                    (swift(K,N, sigma_zero,0.08), 0.08),
                                    (swift(K,N, sigma_zero,0.09), 0.09),
                                    (swift(K,N, sigma_zero,0.1), 0.1),
                                    (swift(K,N, sigma_zero,0.11), 0.11),
                                    (swift(K,N, sigma_zero,0.12), 0.12),
                                    (swift(K,N, sigma_zero,0.13), 0.13),
                                    (swift(K,N, sigma_zero,0.14), 0.14),
                                    (swift(K,N, sigma_zero,0.15), 0.15),
                                    (swift(K,N, sigma_zero,0.16), 0.16),
                                    (swift(K,N, sigma_zero,0.17), 0.17),
                                    (swift(K,N, sigma_zero,0.18), 0.18),
                                    (swift(K,N, sigma_zero,0.19), 0.19),
                                    (swift(K,N, sigma_zero,0.2), 0.2),
                                    (swift(K,N, sigma_zero,0.21), 0.21),
                                    (swift(K,N, sigma_zero,0.22), 0.22),
                                    (swift(K,N, sigma_zero,0.23), 0.23),
                                    (swift(K,N, sigma_zero,0.24), 0.24)))

#---------Definition and submition of the Job----------------------

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job-1', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='', type=
    ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)

mdb.jobs['Job-1'].submit(consistencyChecking=OFF)
mdb.jobs['Job-1'].waitForCompletion()

#--------------Definition for the for cycle------------------------

sigma_zero = [80.0, 90.0, 100.0, 110.0, 120.0, 130.0, 140.0, 150.0,
              160.0, 170.0, 180.0, 190.0, 200.0, 210.0, 220.0, 230.0,
              240.0, 250.0, 260.0, 270.0, 280.0, 290.0, 300.0]

valor_n = [0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24, 0.28, 0.3]

valor_k = [280.0, 290.0, 300.0, 310.0, 320.0, 330.0, 340.0, 350.0, 360.0,
           370.0, 380.0, 390.0, 400.0, 410.0, 420.0, 430.0, 440.0, 450.0,
           460.0, 470.0, 480.0, 490.0, 500.0, 510.0, 520.0, 530.0, 540.0,
           550.0, 560.0, 570.0, 580.0, 590.0, 600.0, 610.0, 620.0, 630.0,
           640.0, 650.0, 660.0, 670.0, 680.0, 690.0, 700.0 ]


#---------Creation the for cycle for information----------------
for sigma_zero in sigma_zero:
    for N in valor_n:
        for K in valor_k:
```

```
170             mdb.models['Model-1'].materials['Steel'].Plastic(table=((sigma_zero, 0.0),
171                                     (swift(K,N ,sigma_zero,0.01), 0.01),
172                                     (swift(K,N, sigma_zero,0.02), 0.02),
173                                     (swift(K,N, sigma_zero,0.03), 0.03),
174                                     (swift(K,N, sigma_zero,0.04), 0.04),
175                                     (swift(K,N, sigma_zero,0.05), 0.05),
176                                     (swift(K,N, sigma_zero,0.06), 0.06),
177                                     (swift(K,N, sigma_zero,0.07), 0.07),
178                                     (swift(K,N, sigma_zero,0.08), 0.08),
179                                     (swift(K,N, sigma_zero,0.09), 0.09),
180                                     (swift(K,N, sigma_zero,0.1), 0.1),
181                                     (swift(K,N, sigma_zero,0.11), 0.11),
182                                     (swift(K,N, sigma_zero,0.12), 0.12),
183                                     (swift(K,N, sigma_zero,0.13), 0.13),
184                                     (swift(K,N, sigma_zero,0.14), 0.14),
185                                     (swift(K,N, sigma_zero,0.15), 0.15),
186                                     (swift(K,N, sigma_zero,0.16), 0.16),
187                                     (swift(K,N, sigma_zero,0.17), 0.17),
188                                     (swift(K,N, sigma_zero,0.18), 0.18),
189                                     (swift(K,N, sigma_zero,0.19), 0.19),
190                                     (swift(K,N, sigma_zero,0.2), 0.2),
191                                     (swift(K,N, sigma_zero,0.21), 0.21),
192                                     (swift(K,N, sigma_zero,0.22), 0.22),
193                                     (swift(K,N, sigma_zero,0.23), 0.23),
194                                     (swift(K,N, sigma_zero,0.24), 0.24)))
195
196             mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
197                 explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
198                 memory=90, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
199                 multiprocessingMode=DEFAULT, name='Job-1', nodalOutputPrecision=SINGLE,
200                 numCpus=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='', type=
201                 ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)
202
203             mdb.jobs['Job-1'].submit(consistencyChecking=OFF)
204             mdb.jobs['Job-1'].waitForCompletion()
205
206             #Abrir o odb objecto
207             odb_Path = 'Job-1.odb'
208             dog_ODB = session.openOdb(name=odb_Path)
209
210             #numero de frames
211             frames = len(dog_ODB.steps['Step-1'].frames)
212
213             #numero de nos
214             nodes =
                    len(dog_ODB.steps['Step-1'].frames[1].fieldOutputs['S'].values[0].instance.nodes)
215
216             #numero de elementos
217             elements =
                    len(dog_ODB.steps['Step-1'].frames[1].fieldOutputs['S'].values[0].instance.elements)
218
219             #field output falues
220             no_of_field_output_ut_values =
                    len(dog_ODB.steps['Step-1'].frames[0].fieldOutputs['S'].values)
221
222             import csv
223             with open('dogbone_%s_%s_%s.csv' %(sigma_zero, N, K), 'wb') as file:
224                 writer = csv.writer(file)
225
226
227                 for k in range(frames): #frames
228                     data = []
229                 #ler a forca no reference point
230                     forca =
                            abs(dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[0].data[1]
                            +
```

```
231                             dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[1].data[1]
                                    +
232                             dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[2].data[1]
                                    +
233                             dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[3].data[1]
                                    +
234                             dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[4].data[1])
235
236                 for i in range(no_of_field_output_ut_values): #inserir
                        no_of_field_output_ut_values
237
238                     node_displacement =
                            dog_ODB.steps['Step-1'].frames[k].fieldOutputs['S'].values[i].data[0]
239
240                     #strains
241                     strainx =
                            dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[0]
242                     strainy =
                            dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[1]
243                     strainxy =
                            dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[3]
244
245                     data.append(forca)
246                     data.append(strainx)
247                     data.append(strainy)
248                     data.append(strainxy)
249
250                 writer.writerow(data)
251
252         dog_ODB.close()
253
254         print(sigma_zero)
255         print(N)
256         print(K)
```

A.2 Python - Dogbone Test

A.2.1 Function Get_data_dogbone

```
1  import pandas as pd
2  import numpy as np
3
4  """Function to get data from the csv file, it also gets the material parameters
5  for the Y_training"""
6
7  def get_data(file):
8
9      X_train = []
10     Y_train = []
11
12     #Data from test
13     X_train=pd.read_csv(file, sep=',',header=None)
14
15     #test parameters data
16     id = [i for i in range(len(file)) if file.startswith('_', i)]
17     idp = [i for i in range(len(file)) if file.startswith('.', i)]
18     sigma = float(file[id[1]+1:id[2]])
19     n_value = float(file[id[2]+1:id[3]])
20     k_value = float(file[id[3]+1:idp[len(idp)-1]])
21     par = [sigma, n_value, k_value]
22     Y_train.append(par)
23
24     X_data = X_train.values.ravel()
```

```
    X_data = X_data.reshape(1,8400)

    Y_train = np.array(Y_train)
    Y_train = np.ravel(Y_train)
    Y_train = Y_train.reshape(1,3)

    return X_data, Y_train
```

A.2.2 Training and Prediction of Neural Network

```
import numpy as np
import glob
import os
from get_data import get_data

# the path to your csv file directory
mycsvdir = r'C:\Users\henriquemiguel\Desktop\Tese\dogbone_data'

# get all the csv files in that directory (assuming they have the extension .csv)
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

"""For cycle for retriving all the information on all the csv files aka abaqus
 tests performed, it performs some tasks for the correct dimensionon of the
 matrix"""

x_train = np.empty((len(csvfiles), 8400))
y_train = np.empty((len(csvfiles), 3))

for i, file in enumerate(csvfiles):

    x_data = get_data(file)[0]
    x_train[i,:] = x_data
    print(file)
    #print('Dimenso de X_train:', x_train.shape)

    y_data = get_data(file)[1]
    y_train[i,:] = y_data
    #print('Dimenso de Y_train:', y_train.shape)

## Standardizing dataset prior to training
#from sklearn import preprocessing
from sklearn.preprocessing import MaxAbsScaler

scaler_x = MaxAbsScaler().fit(x_train)
scaler_y = MaxAbsScaler().fit(y_train)

x_train_scaled = scaler_x.transform(x_train)
y_train_scaled = scaler_y.transform(y_train)

#  Criao  da rede
from tensorflow.keras import layers
from tensorflow import keras

#Layer de entrada
inputs = keras.Input(shape=(8400,))

#Hidden Layer n1
dense = layers.Dense(5000,
                     activation='tanh',
                     use_bias=True,
                     bias_initializer='ones',
                     kernel_initializer='orthogonal')
x = dense(inputs)

```

```
#Hidden Layer n2
x = layers.Dense(1500,
                  activation='tanh',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='variance_scaling')(x)

#Hidden Layer n3
x = layers.Dense(5500,
                  activation='tanh',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='truncated_normal')(x)

#Hidden Layer n4
x = layers.Dense(1500,
                  activation='tanh',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='truncated_normal')(x)
                  #Hidden Layer n4
x = layers.Dense(5000,
                  activation='selu',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='variance_scaling')(x)

#Output layer
outputs = layers.Dense(3)(x)

model = keras.Model(inputs=inputs, outputs=outputs, name ='model')

model.compile(loss=keras.losses.MeanSquaredError(),
              optimizer=keras.optimizers.Adam(learning_rate=1.1775e-5),
              metrics=['MeanSquaredError', 'MAE'])

history = model.fit(x_train_scaled,
                  y_train_scaled,
                  epochs=160,
                  shuffle = True,
                  validation_split = 0.2)

val_acc_per_epoch = history.history['mean_squared_error']
best_epoch = val_acc_per_epoch.index(min(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))

val_acc_per_epoch_MAE = history.history['MAE']
best_epoch_MAE = val_acc_per_epoch_MAE.index(min(val_acc_per_epoch_MAE)) + 1
print('Best epoch: %d' % (best_epoch,))

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(history.history['mean_squared_error'])
ax.set(xlabel='Number of epoch',
       ylabel='MSE',
       title='Best number of epochs')
fig.savefig("MSE_best_epochs_143_nise.png")
plt.show()

fig, ax = plt.subplots()
ax.plot(history.history['MAE'])
ax.set(xlabel='Number of epoch',
       ylabel='MAE',
       title='Best number of epochs')
fig.savefig("MAE_best_epochs_143_noise.png")
plt.show()

```

```
history = model.fit(x_train_scaled,
                    y_train_scaled,
                    epochs = best_epoch,
                    shuffle = True,
                    validation_split = 0.2)

mycsvdir = r'C:\Users\henriquemiguel\Desktop\Tese\dogbone_vali'
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

x_test = np.empty((len(csvfiles), 8400))
y_test = np.empty((len(csvfiles), 3))

for i, file in enumerate(csvfiles):

    x_test = get_data(file)[0]
    x_test_scaled = scaler_x.transform(x_test)
    predictions = model.predict(x_test_scaled, batch_size=256, verbose=0)
    preditction = scaler_y.inverse_transform(predictions)
    print(preditction)

from matplotlib import pyplot as plt

plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model Validation Mean Square Error')
plt.ylabel('Mean squared error')
plt.xlabel('Epoch')
plt.legend(['MSE_train', 'MSE_val'], loc='upper right')
plt.show()
plt.savefig('history_plot_MSE_noise.png')

plt.plot(history.history['MAE'])
plt.plot(history.history['val_MAE'])
plt.title('Model Validation MAE')
plt.ylabel('MAE')
plt.xlabel('Epoch')
plt.legend(['MAE_train', 'MAE_val'], loc='upper right')
plt.show()
plt.savefig('history_plot_MAE_noise.png')

from sklearn.model_selection import learning_curve
from keras.wrappers.scikit_learn import KerasRegressor
from matplotlib import pyplot as plt

train_sizes = [1, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6592]

train_sizes, train_scores, validation_scores = learning_curve(
                              estimator = KerasRegressor(build_fn = lambda: model),
                              X = x_train_scaled,
                              y = y_train_scaled,
                              train_sizes = train_sizes,
                              cv = 5,
                              scoring = 'neg_mean_squared_error',
                              shuffle = True)

print('Training scores:\n\n', train_scores)
print('\n', '-' * 70) # separator to make the output easy to read
print('\nValidation scores:\n\n', validation_scores)

import pandas as pd
train_scores_mean = -train_scores.mean(axis = 1)
validation_scores_mean = -validation_scores.mean(axis = 1)
print('Mean training scores\n\n', pd.Series(train_scores_mean, index = train_sizes))
print('\n', '-' * 20) # separator
print('\nMean validation scores\n\n',pd.Series(validation_scores_mean, index = train_sizes))
plt.style.use('seaborn')
plt.plot(train_sizes, train_scores_mean, label = 'Training error')
```

```
plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
plt.ylabel('MSE', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves with learning rate: 1.1775e-5', fontsize = 18, y = 1.03)
plt.legend()
plt.savefig('learning_curve_1,1775e-5')
```

A.2.3 Addition of Noise

```
import glob
import os.path
import csv
from numpy.core.records import array
import pandas as pd
import random
import numpy as np

# the path to your csv file directory
mycsvdir = r'C:\Users\henriquemiguel\Desktop\Tese\test_dat'
save_path = r'C:\Users\henriquemiguel\Desktop\Tese\test_dat'

# get all the csv files in that directory (assuming they have the extension .csv)
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

files = len(csvfiles)
for i, file in enumerate(csvfiles):

    #Determinar o nome do ficheiro:
    id = [i for i in range(len(file)) if file.startswith('d', i)]
    idp = [i for i in range(len(file)) if file.startswith('.', i)]
    name_file = file[id[1]:idp[len(idp)-1]]
    #Reformular o nome do ficheiro para Rogbone_80.0_0.1_280.0
    str_list = list(name_file)
    str_list[0] = 'R'
    new_name = "".join(str_list)

    #Define uma matriz vazia e abre o ficheiro csv;
    rows = []
    csv_file = csv.reader(open(file))

    for row in csv_file:
        row = [float(x) for x in row]
        rows.append(row)
        #print(rows)

    #Determina um number random entre 0-5% para adicionar ruido;
    RNG_generator = np.random.normal(0, 1, len(row))
    #print(RNG_generator)

    #Abrir um documento excell no local novo
    f = open(save_path + '\\' + new_name + '.csv', 'w', newline='')
    writer = csv.writer(f)

    new_rows = []
    new_value = []
    for i in range(len(rows)):
        line = rows[i]
        RNG_generator = np.random.normal(0, 1, len(row))
        value = line + RNG_generator
        writer.writerow(value)
        new_value.append(value)
        new_rows.append(new_value)

    #writer.writerows(new_rows)
```

```
56
57      print('Faltam: ' + str(ficheiros))
58      ficheiros = ficheiros - 1
```

Appendix B

Source Code Cruciform

B.1 Abaqus - Cruciform Test

```
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from odbAccess import *
from abaqus import *
from abaqusConstants import *
import visualization

import sys
#sys.path.append('C:\Users\kevin\Desktop')
from pyD import new

#--------------Definition of some constants-----------------------
doe = new([120.0, 0.175, 520.0, 1.0, 1.0, 1.0],[200.0, 0.275, 610.0, 2.5, 2.5, 2.5],6,6000)


K = 280.0
sigma_zero = 90.0
N = 0.26

Young_modulos = 210000
poisson = 0.3

r0 = 1.680
r45 = 1.890
r90 = 2.253

#--------------Definition of Swift Law-----------------------------
def swift(valor_k,valor_n, sigma_zero, def_plast):

    def_zero = (sigma_zero / valor_k) ** (1./valor_n)

    sigma = valor_k * (def_zero + def_plast) ** valor_n

```

```
45      #print (valor_k, valor_n, sigma_zero, def_plast, def_zero, sigma)
46
47      return sigma
48
49  def r22(r0, r90):
50
51      r22 = sqrt((r90*(r0+1))/(r0*(r90+1)))
52      #print (r22)
53
54      return r22
55
56  def r33(r0, r90):
57
58      r33 = sqrt((r90*(r0+1))/((r0+r90)))
59      #print(r33)
60
61      return r33
62
63  def r12(r0, r45, r90):
64
65      r12 = sqrt((3*r90*(r0+1))/((2*r45+1)*(r0+r90)))
66
67      return r12
68
69
70
71  #--------------Definition of the part-----------------------------
72
73  mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=200.0)
74  mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
75      0.0, 30.0))
76  mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
77      False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[2])
78  mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 30.0), point2=(
79      15.0, 30.0))
80  mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
81      addUndoState=False, entity=
82      mdb.models['Model-1'].sketches['__profile__'].geometry[3])
83  mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
84      addUndoState=False, entity1=
85      mdb.models['Model-1'].sketches['__profile__'].geometry[2], entity2=
86      mdb.models['Model-1'].sketches['__profile__'].geometry[3])
87  mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
88      30.0, 0.0))
89  mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
90      addUndoState=False, entity=
91      mdb.models['Model-1'].sketches['__profile__'].geometry[4])
92  mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
93      addUndoState=False, entity1=
94      mdb.models['Model-1'].sketches['__profile__'].geometry[2], entity2=
95      mdb.models['Model-1'].sketches['__profile__'].geometry[4])
96  mdb.models['Model-1'].sketches['__profile__'].Line(point1=(30.0, 0.0), point2=(
97      30.0, 15.0))
98  mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
99      False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[5])
100 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
101     addUndoState=False, entity1=
102     mdb.models['Model-1'].sketches['__profile__'].geometry[4], entity2=
103     mdb.models['Model-1'].sketches['__profile__'].geometry[5])
104 mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(center=(
105     15.0, 15.0), point1=(15.0, 22.5))
106 mdb.models['Model-1'].sketches['__profile__'].RadialDimension(curve=
107     mdb.models['Model-1'].sketches['__profile__'].geometry[6], radius=7.0,
108     textPoint=(-20.7033576965332, 20.8900489807129))
109 mdb.models['Model-1'].sketches['__profile__'].Line(point1=(30.0, 15.0), point2=
110     (22.0, 15.0))
111 mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
```

```
112     addUndoState=False, entity=
113     mdb.models['Model-1'].sketches['__profile__'].geometry[7])
114 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
115     addUndoState=False, entity1=
116     mdb.models['Model-1'].sketches['__profile__'].geometry[5], entity2=
117     mdb.models['Model-1'].sketches['__profile__'].geometry[7])
118 mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
119     addUndoState=False, entity1=
120     mdb.models['Model-1'].sketches['__profile__'].vertices[7], entity2=
121     mdb.models['Model-1'].sketches['__profile__'].geometry[6])
122 mdb.models['Model-1'].sketches['__profile__'].Line(point1=(15.0, 30.0), point2=
123     (15.0, 22.0))
124 mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
125     False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[8])
126 mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
127     addUndoState=False, entity1=
128     mdb.models['Model-1'].sketches['__profile__'].geometry[3], entity2=
129     mdb.models['Model-1'].sketches['__profile__'].geometry[8])
130 mdb.models['Model-1'].sketches['__profile__'].autoTrimCurve(curve1=
131     mdb.models['Model-1'].sketches['__profile__'].geometry[6], point1=(
132     18.3673095703125, 19.4483222961426))
133 mdb.models['Model-1'].sketches['__profile__'].FilletByRadius(curve1=
134     mdb.models['Model-1'].sketches['__profile__'].geometry[7], curve2=
135     mdb.models['Model-1'].sketches['__profile__'].geometry[9], nearPoint1=(
136     21.9188003540039, 14.9664535522461), nearPoint2=(21.9188003540039,
137     14.4564361572266), radius=2.5)
138 mdb.models['Model-1'].sketches['__profile__'].undo()
139 mdb.models['Model-1'].sketches['__profile__'].FilletByRadius(curve1=
140     mdb.models['Model-1'].sketches['__profile__'].geometry[9], curve2=
141     mdb.models['Model-1'].sketches['__profile__'].geometry[7], nearPoint1=(
142     21.5909881591797, 13.2178220748901), nearPoint2=(23.448600769043,
143     15.1486024856567), radius=2.5)
144 mdb.models['Model-1'].sketches['__profile__'].FilletByRadius(curve1=
145     mdb.models['Model-1'].sketches['__profile__'].geometry[9], curve2=
146     mdb.models['Model-1'].sketches['__profile__'].geometry[8], nearPoint1=(
147     14.4519281387329, 22.179557800293), nearPoint2=(15.1804056167603,
148     22.9445838928223), radius=2.5)
149 mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR, name='Cruciform', type=
150     DEFORMABLE_BODY)
151 mdb.models['Model-1'].parts['Cruciform'].BaseShell(sketch=
152     mdb.models['Model-1'].sketches['__profile__'])
153 del mdb.models['Model-1'].sketches['__profile__']
154
155 #--------------Definition of the mesh-----------------------------
156
157 mdb.models['Model-1'].parts['Cruciform'].seedPart(deviationFactor=0.1,
158     minSizeFactor=0.1, size=2.5)
159 mdb.models['Model-1'].parts['Cruciform'].setElementType(elemTypes=(ElemType(
160     elemCode=CPS4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
161     hourglassControl=DEFAULT, distortionControl=DEFAULT), ElemType(
162     elemCode=CPS3, elemLibrary=STANDARD)), regions=(
163     mdb.models['Model-1'].parts['Cruciform'].faces.getSequenceFromMask((
164     '[#1 ]', ), ), ), ))
165 mdb.models['Model-1'].parts['Cruciform'].generateMesh()
166
167 #--------------Definition of the assembly-----------------------------
168
169 mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
170 mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='Cruciform-1',
171     part=mdb.models['Model-1'].parts['Cruciform'])
172
173
174 #--------------Definition of the step-----------------------------
175
176 mdb.models['Model-1'].StaticStep(name='Step-1', previous='Initial')
177 mdb.models['Model-1'].steps['Step-1'].setValues(initialInc=0.025, maxNumInc=200)
178 mdb.models['Model-1'].steps['Step-1'].setValues(minInc=1e-20)
```

```
179
180
181 #--------------Definition of the boundary conditions-----------------------------
182
183 mdb.models['Model-1'].rootAssembly.Set(edges=
184     mdb.models['Model-1'].rootAssembly.instances['Cruciform-1'].edges.getSequenceFromMask(
185     ('[#100 ]', ), ), name='Set-1')
186 mdb.models['Model-1'].XsymmBC(createStepName='Initial', localCsys=None, name=
187     'XX-SIM', region=mdb.models['Model-1'].rootAssembly.sets['Set-1'])
188 mdb.models['Model-1'].rootAssembly.Set(edges=
189     mdb.models['Model-1'].rootAssembly.instances['Cruciform-1'].edges.getSequenceFromMask(
190     ('[#1 ]', ), ), name='Set-2')
191 mdb.models['Model-1'].YsymmBC(createStepName='Initial', localCsys=None, name=
192     'YY-SIM', region=mdb.models['Model-1'].rootAssembly.sets['Set-2'])
193 mdb.models['Model-1'].rootAssembly.Set(edges=
194     mdb.models['Model-1'].rootAssembly.instances['Cruciform-1'].edges.getSequenceFromMask(
195     ('[#2 ]', ), ), name='Set-3')
196 mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Step-1',
197     distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
198     'XX-DIS', region=mdb.models['Model-1'].rootAssembly.sets['Set-3'], u1=2.0,
199     u2=UNSET, ur3=UNSET)
200 mdb.models['Model-1'].rootAssembly.Set(edges=
201     mdb.models['Model-1'].rootAssembly.instances['Cruciform-1'].edges.getSequenceFromMask(
202     ('[#80 ]', ), ), name='Set-4')
203 mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Step-1',
204     distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name=
205     'YY-DIS', region=mdb.models['Model-1'].rootAssembly.sets['Set-4'], u1=UNSET
206     , u2=2.0, ur3=UNSET)
207
208 #--------------Definition of the Section-----------------------------
209
210 mdb.models['Model-1'].HomogeneousSolidSection(material='Steel', name=
211 'Section-1', thickness=None)
212 mdb.models['Model-1'].parts['Cruciform'].Set(faces=
213     mdb.models['Model-1'].parts['Cruciform'].faces.getSequenceFromMask((
214     '[#1 ]', ), ), name='Set-2')
215 mdb.models['Model-1'].parts['Cruciform'].SectionAssignment(offset=0.0,
216     offsetField='', offsetType=MIDDLE_SURFACE, region=
217     mdb.models['Model-1'].parts['Cruciform'].sets['Set-2'], sectionName=
218     'Section-1', thicknessAssignment=FROM_SECTION)
219
220 #--------------Definition of the Field Output-----------------------------
221
222 mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(variables=(
223 'S', 'PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF'))
224 mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(variables=(
225 'S', 'PE', 'PEEQ', 'PEMAG', 'LE', 'U', 'RF', 'CF', 'TF'))
226 mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(numIntervals=20)
227 #numero de intervalos esta a 20, inicalmente estava a 50
228
229 #--------------Definition of the material-----------------------------
230 """
231 mdb.models['Model-1'].Material(name='Steel')
232 mdb.models['Model-1'].materials['Steel'].Elastic(table=((Young_modulos, poisson), ))
233 mdb.models['Model-1'].materials['Steel'].Plastic(table=((sigma_zero, 0.0),
234                                     (swift(K,N ,sigma_zero,0.01), 0.01),
235                                     (swift(K,N, sigma_zero,0.02), 0.02),
236                                     (swift(K,N, sigma_zero,0.03), 0.03),
237                                     (swift(K,N, sigma_zero,0.04), 0.04),
238                                     (swift(K,N, sigma_zero,0.05), 0.05),
239                                     (swift(K,N, sigma_zero,0.06), 0.06),
240                                     (swift(K,N, sigma_zero,0.07), 0.07),
241                                     (swift(K,N, sigma_zero,0.08), 0.08),
242                                     (swift(K,N, sigma_zero,0.09), 0.09),
243                                     (swift(K,N, sigma_zero,0.1), 0.1),
244                                     (swift(K,N, sigma_zero,0.11), 0.11),
245                                     (swift(K,N, sigma_zero,0.12), 0.12),
```

```
                                    (swift(K,N, sigma_zero,0.13), 0.13),
                                    (swift(K,N, sigma_zero,0.14), 0.14),
                                    (swift(K,N, sigma_zero,0.15), 0.15),
                                    (swift(K,N, sigma_zero,0.16), 0.16),
                                    (swift(K,N, sigma_zero,0.17), 0.17),
                                    (swift(K,N, sigma_zero,0.18), 0.18),
                                    (swift(K,N, sigma_zero,0.19), 0.19),
                                    (swift(K,N, sigma_zero,0.2), 0.2),
                                    (swift(K,N, sigma_zero,0.21), 0.21),
                                    (swift(K,N, sigma_zero,0.22), 0.22),
                                    (swift(K,N, sigma_zero,0.23), 0.23),
                                    (swift(K,N, sigma_zero,0.24), 0.24)))
mdb.models['Model-1'].materials['Steel'].plastic.Potential(table=((1, r22(r0, r90), r33(r0,r90),
     r12(r0, r45, r90), 1, 1), ))
mdb.models['Model-1'].parts['Cruciform'].MaterialOrientation(
    additionalRotationType=ROTATION_NONE, axis=AXIS_3, fieldName='', localCsys=
    None, orientationType=GLOBAL, region=Region(
    faces=mdb.models['Model-1'].parts['Cruciform'].faces.getSequenceFromMask(
    mask=('[#1 ]', ), )), stackDirection=STACK_3)


#--------------Definition and submition of the Job------------------------------

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job-1', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='', type=
    ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)

mdb.jobs['Job-1'].submit(consistencyChecking=OFF)
mdb.jobs['Job-1'].waitForCompletion()
"""
#--------------Definition for the for cycle-----------------

sigma_zero = 160.0

valor_n = [0.175, 0.2, 0.225, 0.25, 0.275]

valor_k = [520.0, 530.0]

import numpy as np
#--------Creation the for cycle for information-----------
for d in doe:

    sigma_zero, N, K, r0, r45, r90 = round(d[0],2), round(d[1],2), round(d[2],2), round(d[3],2),
         round(d[4],2), round(d[5],2)

    mdb.models['Model-1'].Material(name='Steel')
    mdb.models['Model-1'].materials['Steel'].Elastic(table=((Young_modulos, poisson), ))
    mdb.models['Model-1'].materials['Steel'].Plastic(table=((sigma_zero, 0.0),
                                        (swift(K,N ,sigma_zero,0.01), 0.01),
                                        (swift(K,N, sigma_zero,0.02), 0.02),
                                        (swift(K,N, sigma_zero,0.03), 0.03),
                                        (swift(K,N, sigma_zero,0.04), 0.04),
                                        (swift(K,N, sigma_zero,0.05), 0.05),
                                        (swift(K,N, sigma_zero,0.06), 0.06),
                                        (swift(K,N, sigma_zero,0.07), 0.07),
                                        (swift(K,N, sigma_zero,0.08), 0.08),
                                        (swift(K,N, sigma_zero,0.09), 0.09),
                                        (swift(K,N, sigma_zero,0.1), 0.1),
                                        (swift(K,N, sigma_zero,0.11), 0.11),
                                        (swift(K,N, sigma_zero,0.12), 0.12),
                                        (swift(K,N, sigma_zero,0.13), 0.13),
                                        (swift(K,N, sigma_zero,0.14), 0.14),
                                        (swift(K,N, sigma_zero,0.15), 0.15),
                                        (swift(K,N, sigma_zero,0.16), 0.16),
```

```
                                    (swift(K,N, sigma_zero,0.17), 0.17),
                                    (swift(K,N, sigma_zero,0.18), 0.18),
                                    (swift(K,N, sigma_zero,0.19), 0.19),
                                    (swift(K,N, sigma_zero,0.2), 0.2),
                                    (swift(K,N, sigma_zero,0.21), 0.21),
                                    (swift(K,N, sigma_zero,0.22), 0.22),
                                    (swift(K,N, sigma_zero,0.23), 0.23),
                                    (swift(K,N, sigma_zero,0.24), 0.24)))
mdb.models['Model-1'].materials['Steel'].plastic.Potential(table=((1, r22(r0, r90),
     r33(r0,r90), r12(r0, r45, r90), 1, 1), ))
mdb.models['Model-1'].parts['Cruciform'].MaterialOrientation(
    additionalRotationType=ROTATION_NONE, axis=AXIS_3, fieldName='', localCsys=
    None, orientationType=GLOBAL, region=Region(
    faces=mdb.models['Model-1'].parts['Cruciform'].faces.getSequenceFromMask(
    mask=('[#1 ]', ), )), stackDirection=STACK_3)


mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job-1', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='', type=
    ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)

mdb.jobs['Job-1'].submit(consistencyChecking=OFF)
mdb.jobs['Job-1'].waitForCompletion()

#Abrir o odb objecto
odb_Path = 'Job-1.odb'
dog_ODB = session.openOdb(name=odb_Path)

#numero de frames
frames = len(dog_ODB.steps['Step-1'].frames)

#numero de nos
nodes = len(dog_ODB.steps['Step-1'].frames[1].fieldOutputs['S'].values[0].instance.nodes)

#numero de elementos
elements =
     len(dog_ODB.steps['Step-1'].frames[1].fieldOutputs['S'].values[0].instance.elements)

#field output falues
no_of_field_output_ut_values = len(dog_ODB.steps['Step-1'].frames[0].fieldOutputs['S'].values)

import csv
with open('crucif_%s_%s_%s_%s_%s_%s.csv' %(sigma_zero, N, K, r0, r45, r90), 'wb') as file:
    writer = csv.writer(file)


    for k in range(frames): #frames
        data = []
    #ler a forca no reference point
        forca = abs(dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[40].data[1] +
                    dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[79].data[1] +
                    dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[41].data[1] +
                    dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[67].data[1] +
                    dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[23].data[1] +
                    dog_ODB.steps['Step-1'].frames[k].fieldOutputs['TF'].values[39].data[1])


        for i in range(no_of_field_output_ut_values): #inserir no_of_field_output_ut_values

                node_displacement =
                     dog_ODB.steps['Step-1'].frames[k].fieldOutputs['S'].values[i].data[0]

                #node_tension =
                     dog_ODB.steps['Step-1'].frames[frames[k]].fieldOutputs['U'].values[1].instance.nodes[i].label
```

```
374
375                     #strains
376                     strainx = dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[0]
377                     strainy = dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[1]
378                     strainxy =
                            dog_ODB.steps['Step-1'].frames[k].fieldOutputs['PE'].values[i].data[3]
379
380                     #conjunto = (strainx, strainy, strainxy)
381
382                     #data.append(conjunto)
383
384                     data.append(forca)
385                     data.append(strainx)
386                     data.append(strainy)
387                     data.append(strainxy)
388
389             writer.writerow(data)
390
391     dog_ODB.close()
392
393     print(sigma_zero, N, K, r0, r45, r90)
394     print(tamanho)
395     #tamanho = tamanho - 1
```

B.2 Python - Cruciform Test

B.2.1 Function Get_data_cruciform

```
1  import pandas as pd
2  import numpy as np
3
4  """Function to get data from the csv file, it also gets the material parameters
5  for the Y_training"""
6
7  def get_data(file):
8
9      X_train = []
10     Y_train = []
11
12     #Data from test
13     X_train=pd.read_csv(file, sep=',',header=None)
14
15     #test parameters data
16     id = [i for i in range(len(file)) if file.startswith('_', i)]
17     idp = [i for i in range(len(file)) if file.startswith('.', i)]
18     sigma = float(file[id[1]+1:id[2]])
19     n_value = float(file[id[2]+1:id[3]])
20     k_value = float(file[id[3]+1:id[4]])
21     r0 = float(file[id[4]+1:id[5]])
22     r45 = float(file[id[5]+1:id[6]])
23     r90 = float(file[id[6]+1:idp[len(idp)-1]])
24     par = [sigma, n_value, k_value, r0, r45, r90]
25     Y_train.append(par)
26
27     X_data = X_train.values.ravel()
28     X_data_cru = X_data.reshape(1,9576) #numero no sei
29
30     Y_train = np.array(Y_train)
31     Y_train = np.ravel(Y_train)
32     Y_train = Y_train.reshape(1,6)
33
34     return X_data_cru, Y_train
```

B.2.2 Training and Prediction of Neural Network

```
import numpy as np
import glob
import os
from get_data_cruciforme import get_data

mycsvdir = r'D:\henriquemiguel\Tese\Cruciforme\training_cruciforme'

# get all the csv files in that directory (assuming they have the extension .csv)
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

"""For cycle for retriving all the information on all the csv files aka abaqus
 tests performed, it performs some tasks for the correct dimensionon of the
 matrix"""


x_train = np.empty((len(csvfiles), 9576))
y_train = np.empty((len(csvfiles), 6))

for i, file in enumerate(csvfiles):

    x_data = get_data(file)[0]
    x_train[i,:] = x_data
    #print('Dimenso de X_train:', x_train.shape)

    y_data = get_data(file)[1]
    y_train[i,:] = y_data
    #print('Dimenso de Y_train:', y_train.shape)
    print (file)

## Standardizing dataset prior to training
#from sklearn import preprocessing
from sklearn.preprocessing import MaxAbsScaler

scaler_x = MaxAbsScaler().fit(x_train)
scaler_y = MaxAbsScaler().fit(y_train)

x_train_scaled = scaler_x.transform(x_train)
y_train_scaled = scaler_y.transform(y_train)

#  Criao   da rede
from tensorflow.keras import layers
from tensorflow import keras

#Layer de entrada
inputs = keras.Input(shape=(9576,))

#Hidden Layer n1
dense = layers.Dense(7000,
                  activation='tanh',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='orthogonal')
x = dense(inputs)

#Hidden Layer n2
x = layers.Dense(3000,
                  activation='tanh',
                  use_bias=True,
                  bias_initializer='ones',
                  kernel_initializer='truncated_normal')(x)

#Hidden Layer n3
x = layers.Dense(5000,
                  activation='tanh',
```

```
                    use_bias=True,
                    bias_initializer='ones',
                    kernel_initializer='truncated_normal')(x)

#Hidden Layer n4
x = layers.Dense(5000,
                    activation='selu',
                    use_bias=True,
                    bias_initializer='ones',
                    kernel_initializer='variance_scaling')(x)
                    #Hidden Layer n4
x = layers.Dense(6000,
                    activation='selu',
                    use_bias=True,
                    bias_initializer='ones',
                    kernel_initializer='variance_scaling')(x)

#Output layer
outputs = layers.Dense(6)(x)

model = keras.Model(inputs=inputs, outputs=outputs, name ='model')

model.compile(loss=keras.losses.MeanSquaredError(),
              optimizer=keras.optimizers.Adam(learning_rate=1.512e-5),
              metrics=['MeanSquaredError', 'MAE'])

history = model.fit(x_train_scaled,
                    y_train_scaled,
                    epochs=160,
                    shuffle = True,
                    validation_split = 0.2)

val_acc_per_epoch = history.history['mean_squared_error']
best_epoch = val_acc_per_epoch.index(min(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))

val_acc_per_epoch_MAE = history.history['MAE']
best_epoch_MAE = val_acc_per_epoch_MAE.index(min(val_acc_per_epoch_MAE)) + 1
print('Best epoch: %d' % (best_epoch,))

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(history.history['mean_squared_error'])
ax.set(xlabel='Number of epoch',
       ylabel='MSE',
       title='Best number of epochs')
#fig.savefig("MSE_best_epochs_143.png")
plt.show()

fig, ax = plt.subplots()
ax.plot(history.history['MAE'])
ax.set(xlabel='Number of epoch',
       ylabel='MAE',
       title='Best number of epochs')
#fig.savefig("MAE_best_epochs_143.png")
plt.show()

best_epoch = 143
history = model.fit(x_train_scaled,
                    y_train_scaled,
                    epochs = best_epoch,
                    shuffle = True,
                    validation_split = 0.2)

mycsvdir = r'C:\Users\henriquemiguel\Desktop\Tese\valid'
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

```

```
x_test = np.empty((len(csvfiles), 8400))
y_test = np.empty((len(csvfiles), 3))

for i, file in enumerate(csvfiles):

    x_test = get_data(file)[0]
    x_test_scaled = scaler_x.transform(x_test)
    predictions = model.predict(x_test_scaled, batch_size=256, verbose=0)
    preditction = scaler_y.inverse_transform(predictions)
    print(preditction)

from matplotlib import pyplot as plt

plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('Model Validation Mean Square Error')
plt.ylabel('Mean squared error')
plt.xlabel('Epoch')
plt.legend(['MSE_train', 'MSE_val'], loc='upper right')
plt.show()
plt.savefig('history_plot_MSE.png')

plt.plot(history.history['MAE'])
plt.plot(history.history['val_MAE'])
plt.title('Model Validation MAE')
plt.ylabel('MAE')
plt.xlabel('Epoch')
plt.legend(['MAE_train', 'MAE_val'], loc='upper right')
plt.show()
plt.savefig('history_plot_MAE.png')

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Validation Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
#plt.savefig('history_plot_MAE.png')

from sklearn.model_selection import learning_curve
from keras.wrappers.scikit_learn import KerasRegressor
from matplotlib import pyplot as plt

train_sizes = [1, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6592]

train_sizes, train_scores, validation_scores = learning_curve(
                              estimator = KerasRegressor(build_fn = lambda: model),
                              X = x_train_scaled,
                              y = y_train_scaled,
                              train_sizes = train_sizes,
                              cv = 5,
                              scoring = 'neg_mean_squared_error',
                              shuffle = True)

print('Training scores:\n\n', train_scores)
print('\n', '-' * 70) # separator to make the output easy to read
print('\nValidation scores:\n\n', validation_scores)

import pandas as pd
train_scores_mean = -train_scores.mean(axis = 1)
validation_scores_mean = -validation_scores.mean(axis = 1)
print('Mean training scores\n\n', pd.Series(train_scores_mean, index = train_sizes))
print('\n', '-' * 20) # separator
print('\nMean validation scores\n\n',pd.Series(validation_scores_mean, index = train_sizes))
plt.style.use('seaborn')
plt.plot(train_sizes, train_scores_mean, label = 'Training error')
```

```
plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
plt.ylabel('MSE', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves with learning rate: 1.1775e-5', fontsize = 18, y = 1.03)
plt.legend()
plt.savefig('learning_curve_1,1775e-5')
```

B.2.3 Addition of Noise

```
# %%
import glob
import os.path
import csv
from numpy.core.records import array
import pandas as pd
import random
import numpy as np

# the path to your csv file directory
mycsvdir = r'D:\henriquemiguel\Tese\Cruciforme\training_cruciforme'
save_path = r'D:\henriquemiguel\Tese\Cruciforme\training_Noisiforme'

# get all the csv files in that directory (assuming they have the extension .csv)
csvfiles = glob.glob(os.path.join(mycsvdir, '*.csv'))

files = len(csvfiles)
for i, file in enumerate(csvfiles):

    #Determinar o nome do ficheiro:
    id = [i for i in range(len(file)) if file.startswith('c', i)]
    idp = [i for i in range(len(file)) if file.startswith('.', i)]
    name_file = file[id[3]:idp[len(idp)-1]]
    #Reformular o nome do ficheiro para Rrucif_200.0_0.24_529.66_1.93_1.19_1.46.csv
    str_list = list(name_file)
    str_list[0] = 'R'
    new_name = "".join(str_list)

    #Define uma matriz vazia e abre o ficheiro csv;
    rows = []
    csv_file = csv.reader(open(file))

    for row in csv_file:
        row = [float(x) for x in row]
        rows.append(row)
        #print(rows)

    #Determina um number random entre 0-5% para adicionar ruido;
    RNG_generator = np.random.normal(0, 1, len(row))
    #print(RNG_generator)

    #Abrir um documento excell no local novo
    f = open(save_path + '\\' + new_name + '.csv', 'w', newline='')
    writer = csv.writer(f)

    new_rows = []
    new_value = []
    for i in range(len(rows)):
        line = rows[i]
        RNG_generator = np.random.normal(0, 1, len(row))
        value = line + RNG_generator
        writer.writerow(value)
        new_value.append(value)
        new_rows.append(new_value)

```

```
56      #writer.writerows(new_rows)
57
58      print('Faltam: ' + str(files))
59      files = files - 1
60  # %%
```