applied sciences

MDPI

Article

Comparative Study of Various Neural Network Types for Direct Inverse Material Parameter Identification in Numerical Simulations

Paul Meißner *, Tom Hoppe and Thomas Vietor

Institute for Engineering Design, Technische Universität Braunschweig, Hermann-Blenk-Strasse 42, 38108 Brunswick, Germany

* Correspondence: p.meissner@tu-braunschweig.de; Tel.: +49-5313-916-5019

Abstract: Increasing product requirements in the mechanical engineering industry and efforts to reduce time-to-market demand highly accurate and resource-efficient finite element simulations. The required parameter calibration of the material models is becoming increasingly challenging with regard to the growing variety of available materials. Besides the classical iterative optimization-based parameter identification method, novel machine learning-based methods represent promising alternatives, especially in terms of efficiency. However, the machine learning algorithms, architectures, and settings significantly affect the resulting accuracy. This work presents a comparative study of different machine learning algorithms based on virtual datasets with varying settings for the direct inverse material parameter identification method. Multilayer perceptrons, convolutional neural networks, and Bayesian neural networks are compared; and their resulting prediction accuracies are investigated. Furthermore, advantages in material parameter identification by uncertainty quantification using the Bayesian probabilistic approach are examined and discussed. The results show increased prediction quality when using convolutional neural networks instead of multilayer perceptrons. The assessment of the aleatoric and epistemic uncertainties when using Bayesian neural networks also demonstrated advantages in evaluating the reliability of the predicted material parameters and their influences on the subsequent finite element simulations.

Keywords: parameter identification; machine learning; convolutional neural networks; Bayesian neural networks; LS-DYNA; MAT_187_SAMP-1; GISSMO failure model

Citation: Meißner, P.; Hoppe, T.; Vietor, T. Comparative Study of Various Neural Network Types for Direct Inverse Material Parameter Identification in Numerical Simulations. *Appl. Sci.* **2022**, *12*, 12793. https://doi.org/10.3390/app122412793

Academic Editor: Andrea Prati

Received: 14 November 2022
Accepted: 8 December 2022
Published: 13 December 2022

1. Introduction

In today's engineering product development process, computer-aided engineering (CAE) tools are indispensable to further increase product performance while maintaining safety requirements, thereby satisfying increasing market demands. Using finite element (FE) simulations, different designs and features can be evaluated at an early development stage without having to rely on numerous physical prototypes [1]. A precondition for the use of high-precision simulations involves the consideration of the specific material behavior using suitable material models of the corresponding FE solvers [2]. Concerning the continuously increasing available material types and mixtures, a high demand for calibrated material cards thus arises. For the nowadays increasingly used materials with complex characteristics, such as polymers, the identification of appropriate material parameters (MPs) for the simulation is a challenging task. Often, these parameters cannot be measured directly from experiments, and thus a material parameter identification (MPI) process is required [2,3]. These nonstandard processes include trial and error procedures. In order to achieve the highest possible agreement between simulation and reality, the estimation of the MPs in an optimization-based calibration process is necessary. Typically, different optimization algorithms are used to solve these inverse problems. Grabski et al. [4] applied the metaheuristic-based virus optimization algorithm in combination

with meshless methods [5]. In contrast, Jones et al. [2] used the virtual fields method with the gradient-based algorithm fmincon to solve the inverse problem for full field data. Recently, the applicability of machine-learning-based methods is increasingly investigated, which shows promising results [6,7]. These methods mostly aim at reducing the number of computationally intensive calculations.

This paper focuses on the direct inverse MPI method, in which a surrogate model (in this case a neural network) learns the relationship between FE simulation results and the corresponding material parameters [6,8]. The overall goal of this procedure is to pass the experimental result curves as input to the NN and to predict the MPs which can best reproduce the experimentally determined material behavior [6]. The knowledge gained from the simulations is hence stored in terms of the parameters of the NN and can be repeatedly retrieved, allowing for exceedingly efficient reuse.

Yagawa et al. [9] first proposed the use of a feedforward artificial neural network (FFANN) to predict the MP of a material model. In recent years, the direct inverse method has been further investigated and applied to other materials. Abendroth et al. [10] used this approach to identify the plastic deformation and failure properties of ductile steels. The simulated material behavior based on the ductile elastoplastic damage GTN-model of Gurson, Tvergaard, and Needleman and the load-displacement curve of a small punch test were used as NN input. Furthermore, Chamekh et al. [11] successfully used the direct inverse method to predict the anisotropic Hill parameters of an elasto-plastic model. Ktari et al. [12] designed a ring tensile specimen with effective homogeneous stress and strain distribution along the hoop direction via shape optimization and used the direct inverse method to predict the material parameters of the yield curve. Asaadi et al. [6] demonstrated the viability of the direct inverse approach for retrieving fast solutions in MPI, but highlighted the influence of stochastic noise in the experimental data and the potential mismatch between the simulation model and experiment, which can lead to poor results in parameter estimation. Quantifying unavoidable uncertainties and evaluating their impact on the reliability of the simulation of complex components are increasingly becoming focuses of MPI tasks [13]. The use of Bayesian neural networks (BNN) has proven beneficial to account for aleatory and epistemic uncertainties and to counteract NNs tendency to overfit due to BNNs' inherent generalization ability [14–17]. Thereby, the model returns not only one (or more) deterministic MP, but a distribution object from which, for example, statistics such as the standard deviation can be computed by sampling. Unger et al. [18] demonstrated the integration of BNNs into the direct inverse MPI and presented the advantages using a simple linear elastic material model with an exponential softening function.

Since the input data of the direct inverse MPI method are mostly force-displacement, force-time, or similar curves, and thus an ordered set of real values, this task can be seen as a kind of time series classification problem [19]. In this case, however, the input data are not assigned a class label which would be typical for classification but a continuous number (the material parameter), which is typical for regression. Within the time series classification, different 1D convolutional neural network (CNN) architectures have proven to achieve very high prediction accuracies in recent years [20,21]. To further increase their prediction accuracy, data augmentation strategies can be used to increase the size of the training dataset [22,23]. These include random transformation, such as applying random noise to the input data, and time series decomposition methods [24]. Aside from the specific NN algorithm and the input dataset that is available for training, the hyperparameters (HP) of the NN architecture represent a major influencing factor on the achievable accuracy of the direct inverse MPI [25,26].

Since the choice of suitable machine learning (ML) algorithms, architectures, and settings for the direct inverse MPI method is very challenging, a comparative study of the achievable prediction accuracy is presented within this paper. Multilayer perceptrons (MLP) and convolutional and Bayesian neural networks were implemented in a self-developed Python framework, and their performances were compared based on different settings

and datasets. A self-programmed custom loss function (CLF) was also implemented in the framework. The used CNNs are based on custom network architectures and on established architectures from the literature. Using the library Kerastuner, hyperparameter optimizations (HPO) were performed. Additionally, the influences of applying different Gaussian noise components to the training data and empirical mode decomposition as data augmentation techniques were investigated. The data basis of the investigations was a virtually generated dataset derived from experimental investigations of an additively processed acrylonitrile butadiene styrene (ABS). To describe the highly nonlinear material behavior in the FE software LS-DYNA, the material card MAT_187_SAMP-1 (semi-analytical model for polymers) was used in combination with the failure model MAT_ADD_DAMAGE_GISSMO. Thus, the investigations aimed to further improve the overall prediction accuracy of the direct inverse MPI method and to simplify the choice of suitable architectures and settings for researchers and engineers. By varying different methods and settings, the resulting effects and influences were revealed and quantified in different runs.

In Section 2, the methods and NN architectures required for understanding the studies are briefly explained, and the used experimental dataset and the computational studies are described. The results of the study are presented and compared in Section 3. Finally, in Section 4 the results are summarized, and an outlook on possible further investigations is given.

2. Materials and Methods

In the following, the dataset under investigation and the methods and neural network architectures relevant for understanding are explained. For more detailed explanations, reference is made to further literature.

2.1. Virtual Dataset under Investigation

For the comparative study, the LS-DYNA material model MAT_187_SAMP-1 [27,28] combined with the failure model GISSMO [28,29] was chosen, which is suitable to describe specific material characteristics, such as the tension–compression asymmetry and strain-rate dependency, of thermoplastics. The direct inverse MPI method applied is not limited to any particular material model, although the use of such a sophisticated material model allows an extensive investigation and comparison of different architectures and settings due to the complexity of the parameter identification task. As a database for this work, a virtual generated dataset was created by numerical simulations. The underlying material behavior was based on experimental investigations of an additively processed ABS [26] and conforms to the description in [7]. In the following, the continuum mechanical formulations of MAT_187_SAMP-1 relevant to this paper are briefly introduced. For in depth information on MAT_187_SAMP-1 and GISSMO, we refer to [27,29]. For further information regarding a numerical simulation with the commercial solver LS-DYNA, we refer to [30].

Depending on the provided input of the material card, three different yield curve formulations are achievable with MAT_187_SAMP-1. If tension, compression, and shear test data are provided to the material card, the so-called SAMP-1 yields surface definition results:

$$f = \sigma_{vm}^2 - A_0 - A_1 p - A_2 p^2 \leq 0. \tag{1}$$

σ_{vm} represents the von Mises stress and can be formulated as:

$$\sigma_{vm} = \sqrt{\frac{3}{2}[(\sigma_{xx}+p)^2 + (\sigma_{yy}+p)^2 + (\sigma_{zz}+p)^2 + 2\sigma_{xy}^2 + 2\sigma_{yz}^2 + 2\sigma_{xz}^2]}. \tag{2}$$

Here, p is the first stress invariant and can be expressed as:

$$p = -\frac{\sigma_{zz} + \sigma_{yy} + \sigma_{zz}}{3}. \tag{3}$$

Providing the tensile, compression, and shear test data, the unknown constants A_0, A_1, and A_2 can be calculated as functions of the test results:

$$A_0 = 3\sigma_s^2, \tag{4}$$

$$A_1 = 9\sigma_s^2\left(\frac{\sigma_c - \sigma_t}{\sigma_c\sigma_t}\right), \tag{5}$$

$$A_2 = 9\left(\frac{\sigma_c\sigma_t - 3\sigma_s^2}{\sigma_c\sigma_t}\right). \tag{6}$$

Although shear test data were generated in respective virtual experiments, they were not used as input for the material card, but only as a target curve of the parameter identification task and thus as input for the NN. Consequently, only tensile and compression input data were provided to generate the yield surface, which is why the Drucker–Prager yield surface from MAT_187_SAMP-1 was used and the remaining curve was calculated internally from the other two available curves:

$$\sigma_s = \frac{2\sigma_c\sigma_t}{\sqrt{3}(\sigma_t + \sigma_c)}. \tag{7}$$

Depending on the definition of the plastic Poisson's ratio (PPR), MAT_187_SAMP-1 offers the possibility to define an associated or a non-associated flow rule. Since in this work the PPR changing under loading is to be considered in the simulation, the change in PPR is defined as a function of plastic strain, thereby generating the non-associated yield rule. The yield rule is given by:

$$g = \sqrt{q^2 + \alpha p^2}, \tag{8}$$

with α being the angle between the hydrostatic axis and plastic potential and thus a function of plastic strain. By the following formula, α can be obtained:

$$\alpha = \frac{9}{2} \cdot \frac{1 - 2\nu_p}{1 + \nu_p}. \tag{9}$$

For strain-rate dependency, the load curve defining the yield stress in uniaxial tension is replaced by a table definition, assuming dynamic test data are available. Thus, the table contains multiple load curves corresponding to the respective plastic strain rate. In MAT_187_SAMP-1, these rate effects are assumed to be similar to tension for compression and shear. For a more detailed description, please refer to [27,28].

For consideration of failure in the numerical simulations, the failure model generalized incremental stress-state dependent damage model (GISSMO) was applied within this work. It allows incremental damage accumulation and defining an arbitrary triaxiality-dependent failure strain; the latter is the input of the material model. GISSMO permits an incremental description of damage accumulation, including softening and failure, and provides the advantage of specifying an arbitrary triaxiality-dependent failure strain. The incremental damage accumulation is defined by [28,29,31–33]:

$$\dot{D} = \frac{n}{\epsilon_f} D^{(1-\frac{1}{n})} \dot{\epsilon_p}. \tag{10}$$

Here, D represents the current value of damage, $\dot{\epsilon_p}$ is the plastic strain rate, n is the damage exponent, and ϵ_f is the equivalent plastic strain at failure. The forming intensity parameter F takes into account the onset of necking.

$$\dot{F} = \frac{n}{\epsilon_{crit}(\eta)} F^{(1-\frac{1}{n})} \dot{\epsilon^p}, \tag{11}$$

and the critical strain curve $\epsilon_{crit}(\eta)$ is also a function of the triaxiality for coupling damage and stress under proportional loading. The functions D and F primarily differ in the type of limiting strain, which depends on the applied triaxiality forms ϵ_f or $\epsilon_{crit}(\eta)$. Within this formulation, the damage is coupled to the stress tensor. By applying the concept of effective stress of Lemaitre [34,35], once instability is reached, $F = 1$.

$$\sigma_{eff} = \sigma\left(1 - \left(\frac{D - D_{crit}}{1 - D_{crit}}\right)^{m}\right). \tag{12}$$

Here, D_{crit} represents an indicator for reaching the onset of necking. The fading exponent m is used for the regularization of fracture strain and the energy consumed during post-instability deformation [29]. For more detailed information on GISSMO, please refer to [29].

In several material cards, and so in MAT_187_SAMP-1, there exists the possibility to import experimental data directly as a defined curve or a table. Since these experimentally measured data usually do not lead to the desired agreement or sometimes cannot even be measured completely, the material card input curves (MCIC) are parameterized by a mathematical formulation, and subsequently their parameters are calibrated in an optimization or trial-and-error process. Thus, the MP to be determined are the parameters of the mathematical formulation and therefore are only included indirectly into the material card. In the following, the formulation of the input consisting of parameters and curves used in this work is briefly presented. This is adapted from the work in [7], but the 19 material parameters used were reduced by the bulk modulus, since this exerts no influence on the used shell models.

In the present work, the yield curve formula of Meißner et. al [26] with a combination of second-degree polynomial and root function was used:

$$\sigma = a \cdot \epsilon_{pl}^2 + b \cdot \epsilon_{pl} + c + d \cdot \sqrt{\epsilon_{pl}}, \tag{13}$$

with a, b, c, and d representing the material parameters and ϵ_{pl} the plastic strain. The relationship of the PPR as a function of plastic strain in uniaxial tension, and compression is described with the following formula:

$$\nu_p = \nu_{p,plat} - (\nu_{p,plat} - \nu_{p,press}) \cdot e^{min\left(\frac{-5 \cdot \epsilon_{pl}}{\epsilon_{p,plat}};0\right)}, \tag{14}$$

with ν_p being the plastic Poisson's ratio. The formula describes an exponential decay from the constant PPR $\nu_{p,press}$ of the compression side to a plateau on the tension side. Here, $\nu_{p,plat}$ is the respective PPR value of the tension side and $\epsilon_{p,plat}$ defines the value where 99% of the difference between compression and tension is subtracted.

For consideration of strain rate dependency, a table with multiple yield curves at different strain rates must be provided [28]. The Cowper Symonds analytical approach [36] was used to scale the quasi-static yield curve in this paper:

The following formulation results when this equation is included in the yield curve expression (13):

$$\sigma = \left(a \cdot \epsilon_{pl}^2 + b \cdot \epsilon_{pl} + c + d \cdot \sqrt{\epsilon_{pl}}\right)\left[1 + \left(\frac{\dot{\epsilon}}{C}\right)^{\frac{1}{P}}\right], \tag{15}$$

with the strain rate $\dot{\epsilon}$ and the material parameters C and P.

Failure is implemented by providing the lcsdg curve in GISSMO, which defines the plastic strain at failure (EPSF) as a function of stress triaxiality. Within this work, this curve is defined by a discontinuous linear function consisting of seven reference points, wherein the ordinate values (EPSF) represent the material parameters (see Table 1).

Table 1. Material parameters *epsf* for description of failure with GISSMO lcsdg.

Reference Point	Stress Triaxiality (-)	Equivalent Plastic Strain at Failure (EPSF) (-)
1	−0.333	$epsf_0$
2	−0.250	$epsf_0$
3	0.050	$epsf_1$
4	0.200	$epsf_2$
5	0.430	$epsf_2$
6	0.666	$epsf_3$
7	0.700	$epsf_3$

This curve is derived from the failure behavior of unreinforced thermoplastics known from the literature [37] and is specified by the four *epsf* parameters. Failure occurs, and simultaneously, the element is deleted when an element reaches an equivalent plastic strain (EPS) greater than or equal to the threshold defined by this curve.

As mentioned initially, the direct inverse MPI method needs artificially created data for training the surrogate model. This requires the definition of a material parameter range in which the solution of the inverse problem is assumed. The material parameters of the virtual experimental dataset (MP_{exp}) used in this work and the corresponding MP ranges and the respective curve description are shown in Table A2. Additionally, the constants and settings used in the material card are listed in Table A1.

For creating the virtual dataset, ten different numerical simulations were performed:

- a quasi-static tensile test;
- four strain rate dependent tensile tests (velocity 1–4);
- a compression test;
- a three-point bending test;
- a punch test;
- and two different shear tests.

For all tests, the resulting force-displacement curves were evaluated, and the PPR-EPS curves exclusively for the compression and punch test using history variables 2 and 27, respectively. Consequently, a total of twelve simulation output curves (SOCs) resulted. The corresponding FE models were built as shell models with ELFORM = 16 and are shown in Figure 1. Using the parameter DT2MS, mass-scaling was applied for all models to reduce computational time. To further reduce computation times, time-scaling was used for all tests except the dynamic. Exclusively, the four dynamic tensile tests V1–V4 were simulated with the strain-rate-dependent material card and in that way are only reported in the simulation results of these four dynamic tests. The applied test velocities and resulting strain rates are listed in Table 2.

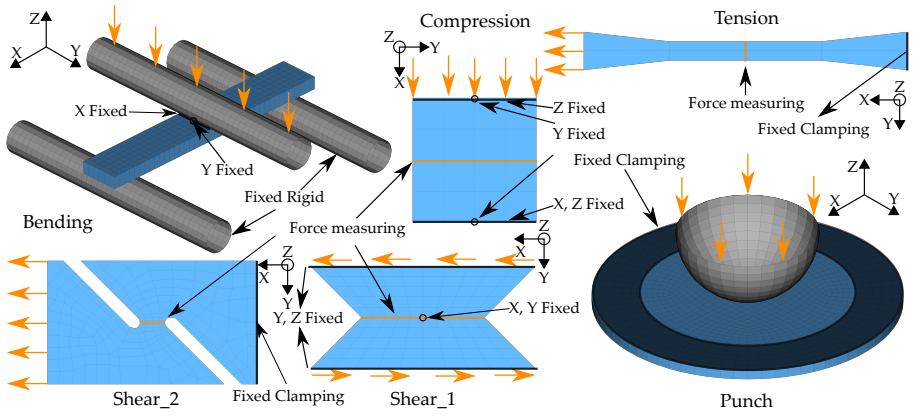


Figure 1. Used shell models [7].

Table 2. Test velocities and resulting strain rates of the virtual dataset.

	V1	V2	V3	V4
Velocity ($\frac{mm}{s}$)	0.0166	0.1666	16.666	1666.6
Strain Rate ($\frac{1}{s}$)	1×10^{-4}	1×10^{-3}	1×10^{-1}	$1 \cdot 10^{1}$

A labeled dataset always consists of the twelve SOCs, which are the input of the NN, its output, and the corresponding material parameters. Besides the virtual experimental dataset, different datasets of varying sizes were created for the comparative study. As usual, these were divided into training and validation datasets to counteract overlearning and to evaluate the performances of the NNs for unseen data. The sampling method used was Latin hypercube sampling (LHS) using the library PyDOE V.0.3.8, which has shown good results in previous work [26]. For datasets 1–4, the sampling was carried out individually for the respective material characteristics (plasticity, failure, etc.) in contrast to datasets 5–8, for which the sampling was carried out on the entire parameter range for comparison purposes. A training/validation split of approximately 70/30 was used for all datasets. The distribution of the eight datasets is listed in Table 3.

Table 3. Datasets used for the comparative study.

Dataset	Sampling Method	Sampling Type	Training Set Size	Validation Set Size	Complete Set Size
1	LHS	individually	266	114	380
2	LHS	individually	525	225	750
3	LHS	individually	1050	450	1500
4	LHS	individually	2100	900	3000
5	LHS	complete	266	114	380
6	LHS	complete	525	225	750
7	LHS	complete	1050	450	1500
8	LHS	complete	2100	900	3000

2.2. Direct, Inverse Neural Network-Based Material Parameter Identification Process

In combination with a numerical model being able to adequately reproduce reality, the MPI has a major influence on structure modeling and sustainability assessment. In addition to the trial-and-error approach, the MPI process consists of solving an inverse identification problem to identify or estimate the system parameters based on the measured system response [38]. For solving this identification method, two general methods are distinguished (see Figure 2). In the most commonly used iterative optimization-based method, an error function is defined as the difference between experimental measurements and parameterized model outputs. This error is usually minimized using an optimization algorithm (e.g., gradient-based or genetic algorithm [39,40]). Since no standardized procedure or terminology exists, this method is called, e.g., by Kučerová [41], the forward (classical) mode. With the expanding application of artificial intelligence (AI), a hybrid ML-based method emerged based on the forward mode, wherein time-consuming numerical simulations are replaced by surrogate models such as neural networks, and then the error measure is minimized using an optimization algorithm [42].

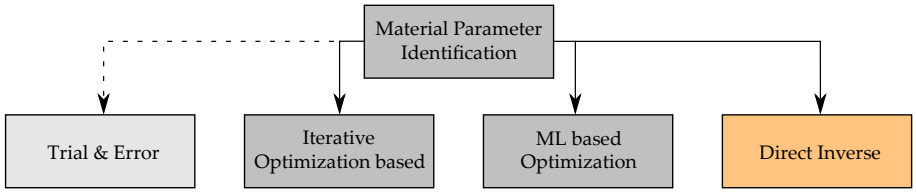


Figure 2. Different methods for material parameter identification.

In the second main approach, the existence of an inverse relationship between output and input was assumed, and by effectively inverting the material model, a surrogate model (e.g., NN) was trained using material responses from finite element analysis as input. Afterwards, the model can be applied to directly predict the model (material) parameters and is therefore called an direct inverse or inverse mode [6,41]. Here, the main advantage is the computationally cost-efficient prediction of the material parameters for repeated MPI, since the generated knowledge is stored in the parameters of the surrogate model and new numerical simulations are not needed [7]. As for the classical forward mode, no standardized procedure for the direct inverse approach exists, and many different methods can be implemented into the process. In the following, the specific Python-based implementation of the direct inverse method used in this work is essentially described (see Figure 3), and reference is made to [7] for a detailed explanation.

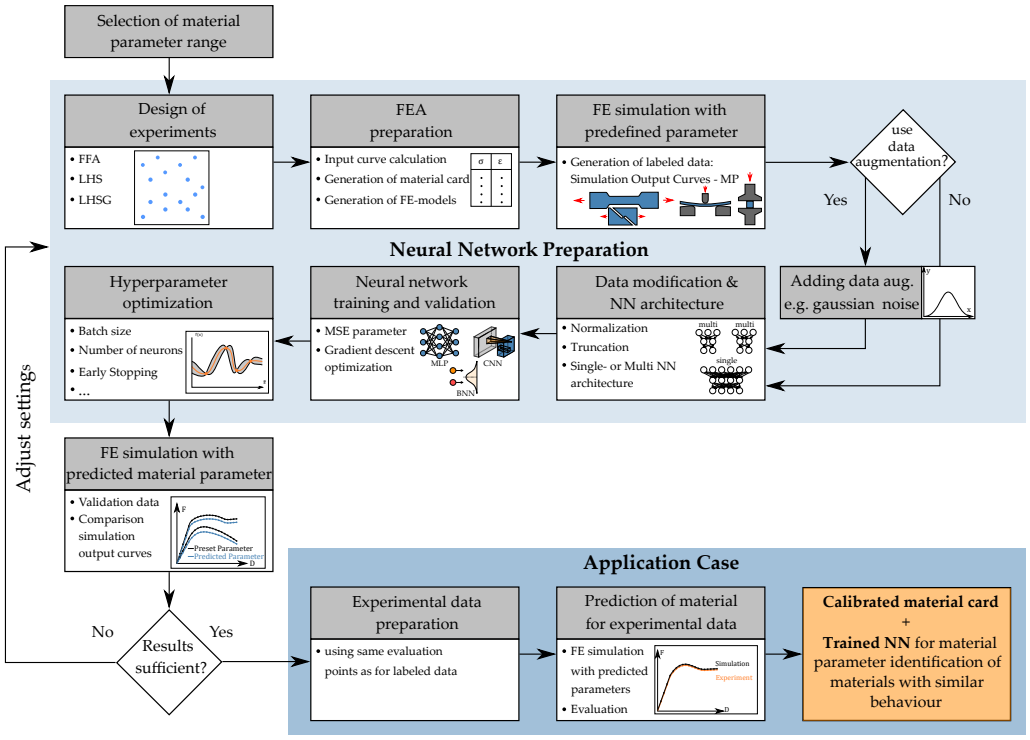


Figure 3. Applied direct inverse material parameter identification method. Own figure based on [7].

In the first step, a material parameter range had to be defined, which ascertained sufficient coverage of the material characteristics. By applying appropriate sampling methods such as LHS, MP sets were generated, and the corresponding material card input curves/tables were generated using the equations presented in Section 2.2. In this work, the resulting system responses comprise the respective force-displacement curves (FDC) and the plastic Poisson's ratio-equivalent plastic strain curves (PEC) of the numerical simulations. For comparability purposes, the FDC and PEC were evaluated by default with an equidistant discretization of 200 points and at the same abscissa locations. Consequently, twelve SOC 2400 input values were opposed to the corresponding 18 material parameters as output. Since the output of the simulations was noise-free and clean, and typically data from experimental measurements are not, Gaussian noise with zero mean and standard deviation of 0.001 was applied to the force and 0.0005 to the PPR data, taking this into account.

As mentioned initially, time series data augmentation (AUG) and manipulation are beneficial for increasing the prediction accuracy of the NN and its generalization ability by reducing overfitting [22,43]. The basic idea of data augmentation is the generation of additional datasets by adding slightly modified variations of already existing data or newly created synthetic data from existing sets while maintaining correct labels [44]. This

is especially relevant for real-world data such as experimental results and can contribute to counteracting problems due to insufficient or expensive generated input datasets. While data AUG is a popular method in image recognition with NNs, it is not a well-established approach for time series regression tasks, and many techniques are borrowed from the former [43]. Many different methods exist, which according to Iwana et al. [24] can be categorized into random transformation, pattern mixing, using generative models (such as NNs), and decomposition. However, not all techniques are beneficial for each time series dataset due to their different features and may also cause prediction-accuracy-degrading effects. In the domain of direct inverse MPI, random transformation in the form of applying Gaussian noise to the time series datasets to account for random measurement errors and deviations has already been successfully applied [7,8,26]. Previously, a noise amount adapted to the noise ratio of the experimental data was used, while the initial dataset size was unchanged. In this work, several noise amounts also were used in some runs to artificially increase the dataset size and to investigate whether an increase in the prediction accuracy can be achieved.

A fundamentally different kind of data augmentation methods are decomposition methods, which decompose time series by extracting features or underlying patterns [24]. Empirical mode decomposition (EMD) is a technique to decompose nonlinear and non-stationary signals. The decomposed oscillatoriy components are typically named an intrinsic mode function (IMF). These, however, differ in nuances depending on the chosen settings of the EMD. In this work, it was applied to the virtual dataset using the Python library PyEMD, which is based on the algorithm by Huang et al. [45]. The input dataset consisting of the simulation results thus was artificially extended by the IMFs resulting from the EMD. As is usual for data augmentation techniques, only the training dataset was increased in size, and for validation, the original data remained unchanged. The chosen settings are explained in Section 2.3. For more detailed information on data augmentation methods, please refer to [23,24,43].

Model-related effects in the numerical simulations, such as oscillations after abrupt failure, complicate the NN training process. Hence, automated data preparation techniques are often required—e.g., truncating the force-displacement curves after failure in the present case. Additionally, filtering functions such as moving averages are applied in this step, which also have to be executed for the prediction based on unknown data. To avoid indirect weighting and problems during the training of the NN, the input and output data were normalized separately. Different ML algorithms can be used as surrogate models, whereas three different NN types were implemented and compared in this work. During the NN training in direct inverse MPI, the mean-squared error (MSE) of the output (material) parameters is usually minimized. In [7], an alternative custom loss function was presented, which was also used in this work (Section 2.2.1). To reduce overlearning tendencies, early stopping was used, whereby the training is stopped after a defined number of epochs without improvement of the validation loss. Additionally, checkpointing was applied, ensuring the best network in terms of loss value was used for the prediction of the validation dataset and the virtual experiment. To further increase the prediction accuracy, it is possible to adapt NN settings, including the number of neurons, training algorithms, etc., concerning the available dataset in a hyperparameter optimization. Finally, after a further numerical simulation using the predicted MP, the agreement of the SOC of prediction and target was assessed, which represents the overall goal of the MPI. For the comparison of the partially steep noisy curves with different lengths, the dynamic time warping (DTW) [46–48] distance measurement algorithm of the Python library similaritymeasures V 0.4.4 was used in this work, which calculated the dimensionless minimum Euclidean distance of the respective curve pair.

In the application step, the NN can finally be used to predict MPs for experimental test data, whereas in the present case, the virtually generated experimental dataset was employed. In this step, the advantage of this approach becomes evident, since in the conventional forward MPI, e.g., for a similar material compound with different additives,

the computationally intensive simulations would have to be performed again. In the following, the neural network types implemented in the direct inverse MPI framework and the related specific features are presented.

2.2.1. Multilayer Perceptron

Multilayer perceptrons are a fully connected class of feedforward artificial neural networks, consisting at least of an input layer, a hidden layer, and an output layer. The network topology can consist of single or multiple hidden layers (HL). In most work related to direct inverse MPI, including this paper, a fully connected architecture has been chosen, where each neuron of one layer is connected to each neuron of the adjacent layer. This results in a relatively high number of network parameters to be optimized in training, which allows a precise approximation of highly nonlinear functions. For further information on MLPs, the reader is referred to [49–52].

In the chosen network architecture, the input layer consists of 2400 neurons corresponding to each of the 200 equidistantly distributed abscissa positions of the twelve SOCs. The output layer, on the other hand, consists of 18 neurons representing the corresponding material parameters. The settings used for the respective investigation runs are listed in Section 2.3 and Table A3. Besides the commonly used loss function, which is calculated from the MSE of the MP, the custom loss (CL) function described in Meissner et al. [7] was used in combination with the MSE of the MP. The CL uses the MSE of the curves calculated with the analytical formulas from Section 2.1 instead of the deviation of the MP. Thus, it considers the observation that sometimes significantly deviating MP can lead to similar MCIC, which in turn leads to high agreement of the simulation results. In this work, this custom loss was used in combination with the MSE of the MP to be able to exploit any synergistic effects that may occur. Despite the normalization of the two loss values, they lie on different scales, and therefore, they were multiplied by each other. The combined loss COL was calculated:

$$COL = MSE_{MP} \cdot CL_{MCIC}. \tag{16}$$

2.2.2. Convolutional Neural Networks

CNNs are often applied in the context of machine image processing [53], where the model learns an internal representation of a two- or three-dimensional input. However, in recent years they have been successfully applied in various other research areas [54], such as language processing tasks [55] and time series analysis [56]. In the latter, usually, one-dimensional sequences of data constitute the input. The model learns to extract features from sequences of observations and how to map the internal features to the possible outputs. Convolutional neural networks consist of one or more convolutional layer, followed by a pooling layer, and these units may be repeated arbitrarily several times [49] (see Figure 4). The central processing step is the convolution, which can be understood as applying and sliding a filter of definable size over the receptive fields of each data point in the time series. The repetition of this linear transformation results in a feature map that indicates the intensity with which the feature encoded in the weights of the filter is present over the length of the time series. Then, the values of the feature map are passed through a nonlinear activation function. Applying several filters, e.g., to learn multiple discriminative features, to a time series, results in as many feature maps as filters were applied. The values of the filter depend highly on the investigated dataset [49,57].

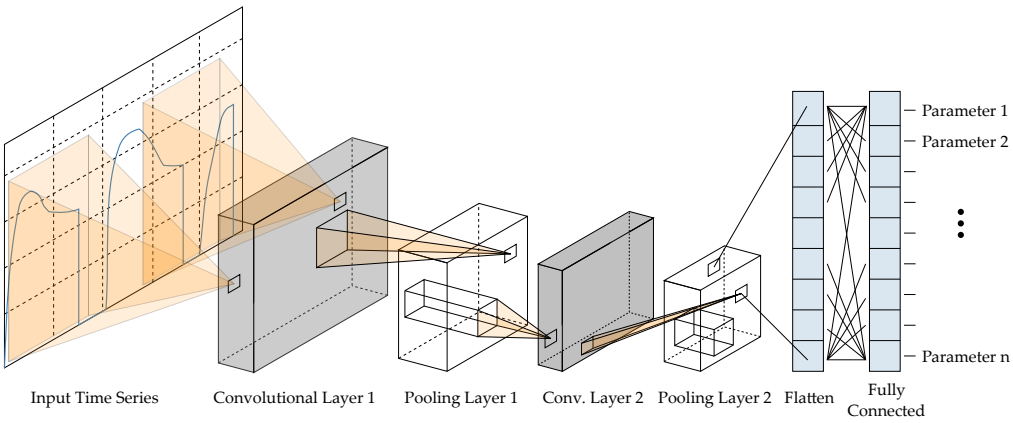


Figure 4. Schematic illustration of a convolutional neural network.

Typically, a pooling layer is used after convolution, which filters out redundant information of the resulting time series by aggregating over a sliding window. Despite the data reduction, the performance of the CNN is usually not reduced, but rather, the computational efficiency is increased, enabling the solving of more complex tasks and also counteracting overlearning. Often, for time series data, additional batch normalization over each channel is applied to enable faster convergence and to keep the mean output close to zero and the output standard deviation near to one. Through a flattened layer, the resulting arrays of pooled feature maps are transformed into a one-dimensional vector. Subsequently, this is fed as input to a fully connected layer (FCL), which resembles the architecture of the MLPs. This can be used for classification or regression, where the number of neurons corresponds to the number of classes or (material) parameters to be determined. Consequently, both feature extraction and regression operations are fused into one process which can be optimized to maximize prediction accuracy. This is an advantage of 1D CNNs, which additionally have comparatively low computational complexity [20,49,57].

Hence, CNNs can learn directly from raw time series data and do not require domain expertise to manually engineer input features. The time series used in this work can be interpreted as one-dimensional grid structures, with data points having local dependencies through the continuously increasing abscissa values (e.g., displacements). The main advantage results from the grid-like structure of the feature maps, which allows one to consider local dependencies in the input in learning the activation patterns. Strong correlations of neighboring data points can be detected by the feature maps and used as activation patterns. The CNNs possess comparatively fewer learnable parameters than MLPs, for which a separate weight is assigned to each input value. The greatest influence on the number of learnable parameters is caused by the width and depth of the subsequent MLP. In contrast, the convolutional layers of the CNN have significantly fewer parameters due to the shared weights and the sparse connections between the neurons [49].

Within this work, various CNN architectures from the literature were implemented in the self-developed framework of direct inverse MPI, and their performances were investigated in the comparative study. Their internal denotation and the corresponding literature sources of the CNNs are listed in Table 4. The CNN proposed by B. Zhao et al. [58] was used as the default architecture for most studies. Based on the network of Wang et al. [59], the two custom architectures FCN2 and FCN3 were included in the experimental plan. In Table A5, the individual settings of different networks are listed in additional to those of Default_CNN by Zhao et al. For detailed information (number of filters in convolution, kernel_size, etc.) about the remaining CNN architectures from the literature, we refer to the corresponding sources in Table 4. Additionally, based on the network of Zhao et al. [58], hyperparameter optimization was performed, and the applied HPO parameter ranges and resulting settings are presented in the Tables A6 and A5.

Table 4. Internal denotation of the CNNs used and corresponding literature sources.

CNN Number	Internal Denotation	Author	Source
1	Default_CNN	B. Zhao et al.	[58]
2	CNN Yang	H. Yang et al.	[60]
3	CNN Azizjon	M. Azizjon et al.	[61]
4	CNN Rautela	M. Rautela et al.	[62]
5	MCDCNN	Y. Zheng et al.	[63]
6	Resnet	Z. Wang et al.	[59]
7	FCN	Z. Wang et al.	[59]
8	Encoder	J. P. Serrà et al.	[64]
9	FCN2	Meißner et al.	-
10	FCN3	Meißner et al.	-

The network of Zhao et al. [58] utilizes two repetitions of a 1D convolutional layer, followed by an average pooling layer. The second pooling layer is followed only by a flattened layer and subsequently by a fully connected dense layer containing the 18 output neurons. Thus, the authors omitted the following MLP. Yang et al. [60] also used a network with two convolutional layers. Additionally, they integrated a dropout layer for each to let random data points become zero during training. Furthermore, they incorporated batch normalization and a subsequent MLP with 32 neurons. Azizjon et al. [61] also applied dropout and batch normalization in their two-layer CNN variant. However, while in Yang et al. the convolution layer is followed by a dropout layer and a pooling layer followed by normalization, in Azizjon et al., two convolutions are performed directly in sequence, followed by the pooling layer with subsequent batch normalization and dropout. By adding a third sequence of convolution and pooling layer, the architecture of Rautela et al. [62] is theoretically capable of more complex feature extraction.

The present dataset is considered to be a multivariate time series (MTS) because the input data consist of the twelve SOCs of the respective simulation results, which depend not only on their previous values but also on further variables. However, different parameters can only be estimated from some SOC. Therefore, it could be advantageous to separate the feature maps of the twelve SOC, which was done within this work by using the multi-channel deep convolutional neural network (MCDCNN) of Zheng et al. [63]. Here, the convolutions are applied independently (in parallel) on each SOC time series. The separate convolutional stages are finally merged by a concatenate layer. Since the architecture applied by Zheng et al. additionally uses a subsequent MLP with 732 neurons, a comparatively high number of learnable parameters resulted. In contrast, the architecture of Wang et al., a residual network (RESNET) [59], again assumes a univariate time series and represents a relatively deep network with eleven layers, of which nine are convolutional layers. The subsequent global average pooling (GAP) layer replaces the flattened layer and averages the time series across the time dimension. This reduces the very high number of otherwise resulting learnable weights and biases. The fully convolutional neural network (FCN) presented by Wang et al. also does not have a flattened layer and uses a GAP layer instead. In contrast to the previous architecture, it only consists of three convolutional blocks and thus is less deep. The encoder CNN used by Serrà et al. [64] was originally inspired by the FCN, the main difference being the replacement of the GAP layer with an attention layer. The attention mechanism enables the network to learn which parts of the time series are responsible for a specific classification or regression part. Like the FCN network, it consists of three convolutional blocks. For detailed information, please refer to the above-mentioned sources.

In the self-adapted FCN2 and FCN3, the complexity of the original network was reduced to counteract overlearning (see Section 3). In contrast to FCN and FCN3, no batch normalization was used for FCN2. Nevertheless, it is mentioned again that the performance of a CNN architecture heavily depends on the fundamental characteristics of the time series, and the CNNs from the literature were used for different time series types.

For the implementation of the networks, the structure of Fawaz et al. [57] was adopted and implemented in the framework. If available, the CNNs implemented in the Fawaz et al. repository were used for the investigation within this work. Based on Default_CNN, hyperparameter optimizations were also performed for some runs, resulting in additional networks (see Table A5).

2.2.3. Bayesian Neural Networks

In most MPI methods, including the direct inverse methods using standard NNs, the output consists of the deterministic material parameters, providing no information about their accuracy and the model's confidence in the prediction. For example, predictions outside the parameter range applied for the training or a too-small training dataset size can result in highly deviating MP predictions, which can lead to a significantly reduced informative quality of subsequent numerical simulations. In this case, information about the existing uncertainties can significantly support the engineer in the assessment of the results and contribute to obtaining correct conclusions from them, e.g., in order to initially increase the prediction accuracy of the method. Furthermore, methods based on NNs often tend to overfit, which limits their generalization ability. One opportunity to overcome these limitations and to maintain prediction accuracy is to use Bayesian neural networks [16], which have the same advantages as the standard neural networks but possess the benefit of providing additional information about the persisting uncertainties.

In the literature, no uniform BNN definition exists, although they can be understood as stochastic artificial neural networks (SANN) which are trained by Bayesian inference [16]. SANN are built by applying stochastic components such as stochastic activation or stochastic weights to the network, to simulate multiple possible models θ with their associated probability distribution $p(\theta)$ [17]. θ are the parameters of the network such as weights and biases, which are determined during the learning process depending on the training data D. Depending on the chosen NN architecture, different BNNs are achievable, such as a Bayesian convolution neural network (BCNN) or Bayesian recurrent neural network (BRNN), whereas in the present work the underlying architecture is an MLP. For BNN model generation, a prior distribution over the possible model parametrization $p(\theta)$ and a prior confidence in the predictive power of the model $p(y|x,\theta)$ has to be chosen, with x being the input and y their corresponding labels. By applying Bayes' theorem and forcing independence between the model parameters and the input, the Bayesian posterior value is given by:

$$p(\theta|D) = \frac{p(D_y|D_x,\theta)p(\theta)}{\int_\theta p(D_y|D_x,\theta')p(\theta')d\theta'} \propto p(D_y|D_x,\theta)p(\theta), \tag{17}$$

with D_x being the training data inputs and D_y the training labels [17]. For further details on the underlying mathematical theory, please refer to [14,16,17,65]. Uncertainties in the prediction of MP can arise due to measurement errors and noise, and insufficient data availability [17]. The former is referred to as aleatoric $p(y|x,\theta)$ and the latter as epistemic $p(\theta|D)$ uncertainty, whereas BNNs provide the further advantage of being able to distinguish between them.

Aleatoric uncertainties lead to unpredictable differences even if a model prediction is repeated for a constant dataset. The uncertainties are inherent in the data, so increasing the size of the training dataset does not reduce these uncertainties. For BNN model generation and implementation in the direct inverse MPI framework, the libraries TensorFlow V.2.1.0, TensorFlow Probability V.0.9.0 (TFP), and Keras V.2.3.1 were used in this work, as for the MLPs and CNNs. To account for aleatoric uncertainties in the model, a TFP IndependentNormal layer was used. Consequently, instead of the deterministic MP, the model returns a distribution object with a learnable mean and standard deviation (STDV). Hence, instead of the MSE or the custom loss function, the negative log-likelihood (NLL) was used, in order to compute how likely it is to get the true data from the estimated distribution. The negative sign is used because minimization is better to apply in computation and

minimizing negative log-likelihood coincides with maximizing the likelihood. To obtain the MP estimates and the standard deviation, we sampled 500 times from the distribution object and calculated the statistics based thereon.

In contrast, epistemic uncertainties can be reduced by adding more data. For implementation into the network, instead of deterministic weights or bias values, weight distributions are learned during the training process (see Figure 5). The preliminary distribution is called the prior, which is then adjusted during the learning process, yielding the posterior distribution. A variety of usable distributions exist, although the choice of a specific prior distribution is not straightforward [17]. The use of a normal distribution with zero mean, a standard deviation of one, and a diagonal covariance is often a suitable default setting [17], which is why this was also used within this work. For the implementation, the TFP DenseVariational layer (DVL) was applied, which allows performing complex calculations, such as the computation of the evidence's lower bound (ELBO). For further details on the mathematical backgrounds and software implementations, we refer to [16,17] and the TensorFlow Probability package. As a consequence of implementing epistemic uncertainties into the network, it returns a different output for each prediction run, since a new set of weights are sampled from the distributions to build the BNN and predict an output. Hence, in the present study, we ran each prediction 100 times to determine the mean values of the outputs and corresponding standard deviations, whereas the latter should decrease when using more data. Due to the large input neuron number of 2400 for the time series, the additional parameters of the network (distribution parameters and covariance matrix) resulted in a very high number of network parameters to be trained, even with a small number of hidden layer neurons. This led to a very long training time or even to untrainable network architectures due to limited hardware resources. An applicable alternative is to utilize only a few or just the last layer as a stochastic one, which may be considered as learning a point estimate transformation, followed by a shallow BNN [17]. For the fully stochastic networks used, the number of points of the SOC was reduced to decrease the input data point amount and thus the network parameter number. In this work, different variants and settings for the BNN were used, which are listed in the Appendix in Table A7.

Figure 5 illustrates a BNN based on an MLP, which can account for both aleatoric and epistemic uncertainties. BNNs have been previously been used for direct inverse MPI for a simple material model by Unger et al. [18], who demonstrated some advantages in the context of MPI. In this work, the BNNs were applied to the proposed complex material models MAT_187_SAMP-1 and GISSMO, and we evaluated the effects of different settings on the prediction accuracy in comparison to other network types. Advantages for the subsequent numerical simulation are also shown, by which the effects of uncertainties can thus be evaluated. Furthermore, it is shown how well the underlying relationship could be learned and how much prediction repetitions or inherent uncertainties affect the prediction result.

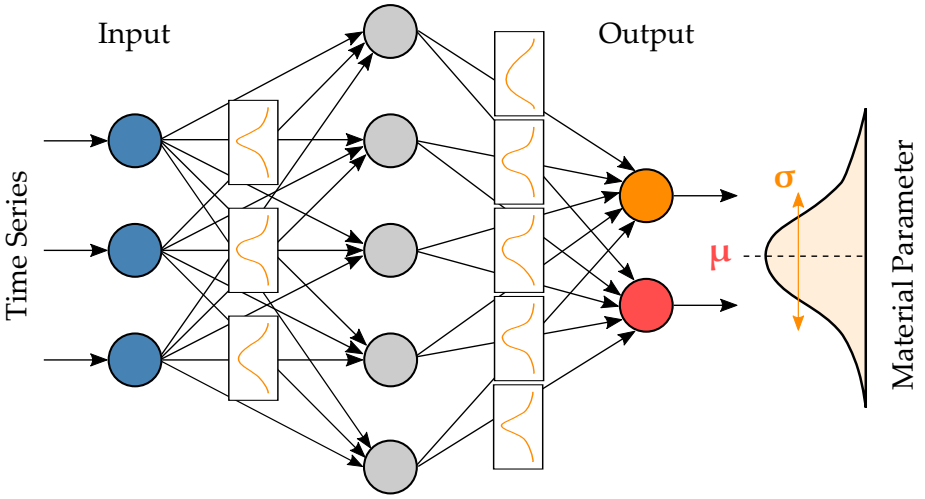


Figure 5. Schematic illustration of a Bayesian neural network.

2.3. Experimental Plan for Comparative Study

The experimental plan included several MPI runs based on the virtual dataset presented in Section 2.1 with various NN architectures and settings (see Table A8). The goal of this MPI is to achieve the highest possible agreement between the simulation results of the various experiments obtained with the MP of the experimental virtual dataset and the corresponding predicted MP. The assessment basis is the dynamic time-warping error averaged for the simulation results; a lower value indicates better agreement. Additionally, this error was also evaluated for the validation dataset, which represents a more representative assessment due to the higher number of samples. In contrast to the experimental dataset, the validation datasets were used to evaluate the prediction accuracy during the training process, which is why the experimental dataset provided additional information regarding the generalization ability of the NN. Besides the DTW error, the values of the loss function were considered for the evaluation of the varying runs, which are also represented the evaluation criterion during the network training. However, only identical loss functions can be compared with each other. Since the numerical simulations necessary for the calculation of the DTW values are very resource-intensive, they were not performed for each run.

As explained in Section 2.2, artificial Gaussian noise was applied to all datasets. For the runs with Noise AUG, the additional Gaussian noise amounts listed in Table A9 were applied to the used unnoised training dataset. Consequently, the training set size of the underlying dataset, dataset 3, increased from 1050 to 5250. For the runs designated as EMD AUG in Table A8, an EMD was performed using the PyEMD V.1.2.3 library so that five additional corresponding IMF curves were added to each SOC. Thus, the training set size of dataset 3 increased from 1050 to 6300.

The presented NN architectures were implemented in the MPI framework with Python V.3.7.7 using TensorFlow V.2.1.0, TensorFlow Probability V.0.9.0, and Keras V.2.3.1. For the hyperparameter optimizations, the library KerasTuner V.1.0.1 was used. The settings of the implemented networks are listed in Appendix A; and the trainings and predictions, were performed on a GPU NVIDIA RTX 3090. For the numerical simulations, the solver LS-DYNA R.11.1 was used.

3. Results and Discussion

The results of the comparative study are listed in Table 5 for the respective runs. In addition to the presented error measures, the training loss of the corresponding epoch with the lowest validation loss was added. As expected, a larger dataset tends to be associated with an increase in prediction accuracy, which can be seen from the decreasing mean DTW distance of the SOC of the validation set. Furthermore, an increase in the accuracy of the DTW distances of the validation sets is often coupled with an increase in the predictive accuracy of the virtual experimental dataset. However, this ratio does not show a linear

relationship, since the error measure of the experimental dataset exhibits high variations due to the sample size of unity. This sample size was chosen because the experimental dataset represents the actual use case of the MPI and therefore consisted of only one set of material parameters. This also demonstrates a fundamental disadvantage of the direct inverse MPI compared to the iterative optimization-based method, since hereby the mean error of a training or validation set was minimized rather than the error of the present MP set. This again emphasizes the importance of the network's generalization capability, which brought both errors closer together on average.

Table 5. Results of comparative study. (* = Not representative, due to insufficient data points on SOCs).

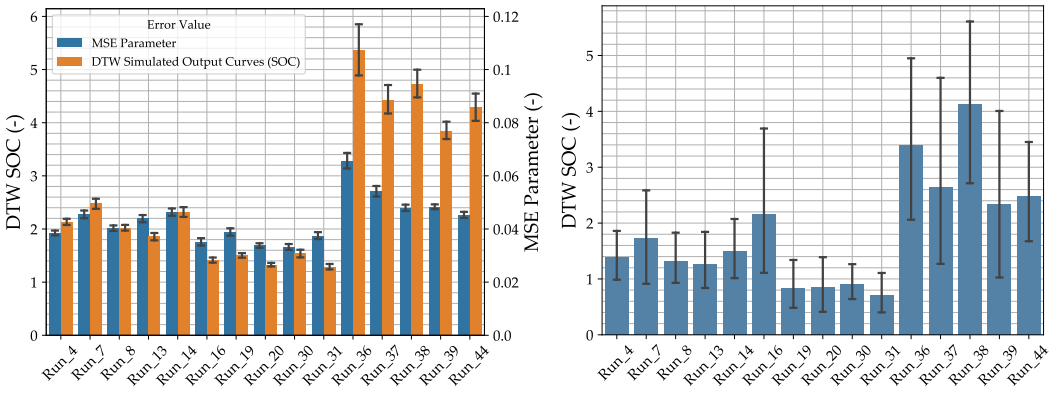
Run	NN	Dataset	Loss	Stopped Epoch	Best Val. Loss (-)	Cor. Training Loss (-)	MSE MP Val. Set (-)	Mean DTW SOC Val. Set (-)	Mean DTW SOC Exp. Set (-)
1	Default_MLP	1	MSE	180	0.0579	0.0370	0.0579	2.835	3.171
2	Default_MLP	2	MSE	184	0.0489	0.0384	0.0489	2.598	2.386
3	Default_MLP	3	MSE	193	0.0433	0.0385	0.0433	2.582	1.505
4	Default_MLP	4	MSE	254	0.0385	0.0333	0.0385	2.134	1.396
5	Default_MLP	1	COL	245	0.00107	0.00049	0.0588	3.174	1.685
6	Default_MLP	2	COL	450	0.00092	0.00046	0.0538	2.779	2.384
7	Default_MLP	3	COL	333	0.00068	0.00036	0.0455	2.477	1.718
8	Default_MLP	4	COL	441	0.00047	0.00027	0.0403	2.024	1.314
9	Default_MLP	5	COL	265	0.00113	0.00071	0.0609	-	-
10	Default_MLP	6	COL	168	0.00089	0.00056	0.0547	-	-
11	Default_MLP	7	COL	310	0.00068	0.00041	0.0467	-	-
12	Default_MLP	8	COL	339	0.00050	0.00037	0.0409	-	-
13	MLPHPO	3	COL	316	0.00059	0.00033	0.0439	1.855	1.257
14	Default_MLP	3	COL	96	0.00064	0.00043	0.0464	2.319	1.494
15	Default_MLP	3	COL	340	0.00128	0.00224	0.0637	4.528	8.485
16	Default_CNN	3	MSE	922	0.0352	0.0285	0.0352	1.416	2.165
17	Default_CNN	1	COL	927	0.00090	0.00044	0.0573	2.413	1.199
18	Default_CNN	2	COL	736	0.00063	0.00039	0.0439	1.660	1.214
19	Default_CNN	3	COL	1000	0.00053	0.00030	0.0388	1.503	0.841
20	Default_CNN	4	COL	946	0.00035	0.00028	0.0338	1.327	0.844
21	CNN_Yang	3	COL	668	0.00144	0.00122	0.0727	12.014	10.555
22	CNN_Azizjon	3	COL	132	0.00086	0.00051	0.0481	3.908	3.835
23	CNN_Rautela	3	COL	152	0.00073	0.00030	0.0469	-	-
24	MCDCNN	3	COL	142	0.00083	0.00037	0.0563	3.308	1.729
25	Resnet	3	COL	238	0.00069	0.00015	0.0451	2.698	2.263
26	FCN	3	COL	174	0.00065	0.00044	0.0443	3.814	2.460
27	Encoder	3	COL	136	0.00082	0.00015	0.0468	2.867	1.660
28	FCN2	3	COL	2000	0.00091	0.00041	0.0655	3.246	3.936
29	FCN3	3	COL	594	0.00048	0.00041	0.0382	2.282	2.505
30	CNNHPO	3	COL	614	0.00045	0.00036	0.0332	1.538	0.91
31	Default_CNN	3	COL	408	0.00048	0.00025	0.0375	1.289	0.699
32	Default_CNN	3	COL	1000	0.00081	0.00185	0.0502	2.985	1.676
33	BNN1	3	NLL	20,000	2.4365	2.7654	0.0706	-	-
34	BNN2	3	NLL	20,000	1.6138	1.8899	0.0703	-	-
35	BNN1	3	NLL	20,000	1.9382	2.2223	0.0668	1.087 *	0.752 *
36	BNN3	1	NLL	5000	1.1575	1.2249	0.0657	5.372	3.865
37	BNN3	2	NLL	5000	−0.1687	−1.8456	0.0542	4.431	2.641
38	BNN3	3	NLL	5000	−2.9159	−3.1798	0.0480	4.734	4.124
39	BNN3	4	NLL	5000	−3.6197	−3.5577	0.0483	3.848	2.334
40	BNN3	3	NLL	5000	−1.1842	1.7050	0.0678	-	-
41	BNN3	3	NLL	5000	−2.1214	−3.3050	0.0501	-	-
42	BNN3	3	NLL	5000	−1.3211	−1.5385	0.0568	-	-
43	BNN3	3	NLL	5000	−2.1592	−2.3138	0.0525	-	-
44	BNN3	3	NLL	5000	−3.7239	4.7501	0.0452	4.284	2.483
45	BNN3	3	NLL	5000	1.8077	2.0721	0.0720	-	-

Based on the validation loss and the corresponding MSE of the material parameters of runs 5 to 12, it can be seen that for the investigated dataset, the sampling types individually or complete of the selected LHS do not have a significant influence on the prediction accuracy. Using the newly defined custom loss function (COL), the prediction accuracy could be increased for the larger datasets, 3 and 4, concerning the DTW distance of the SOC of the validation set. However, this does not apply to the smaller datasets, and therefore, the performance of COL should be investigated more comprehensively in further studies. In general, across all network types, a tendency can be observed that more epochs are needed for training when larger datasets are available, which may result from the network's ability to adjust its parameters more precisely due to a higher number of epochs.

As expected, the MLP hyperparameter optimization also led to an increase in prediction accuracy. In contrast, for the CNN, this only led to slight minimization of the validation loss and the MSE of the MP, along with a simultaneous minor increase in the DTW distances of the SOC. A reason could be that the hyperparameters and network architecture of the Default_CNN are already very well suited for the present dataset. Furthermore, the data augmentation with additional noise amounts was successful, which can be seen in the reductions in the DTW distances of runs 14 and 31 compared to the corresponding runs 7 and 19, which each contained the same settings and no additional noise amounts. This is valid for the DTW distance of both the validation set and the experimental set, providing a significant advantage for the final application of direct inverse MPI on experimental datasets. Run 14, with additional noise amounts and Default_CNN of Zhao et al. [58], achieved the best results of the entire study, despite not being trained on the largest dataset, 4.

However, data augmentation with empirical mode decomposition did not prove to be effective. The reason might have been the implementation into their MPI framework. Due to the widely varying characteristics of the individual time series (see Figure A1), they would have to be decomposed into different numbers of IMFs. This quantity of the resulting IMFs is furthermore dependent on the chosen stopping criteria of the used algorithm, which can be selected separately for different time series. For the present study, a limitation was defined by a maximum number of five resulting IMFs. Without this limitation, certain complex SOCs with sophisticated characteristics, such as the FDC of the tensile test V4, would have to be decomposed into additional curves. Other time series, such as the PEC of the compression and punch test, resulted in a smaller number of IMFs. However, the current implementation in the direct inverse MPI required an equal number of IMFs for the different tests, and therefore, the dataset was replenished with the available IMFs. This resulted in implicit weighting, which may influence the training process. Furthermore, different settings of the EMD algorithm may also affect it, which is why the suitability of data decomposition methods and EMD in particular for increasing the prediction accuracy for complex direct inverse MPIs cannot be conclusively evaluated. This should be further investigated in future studies.

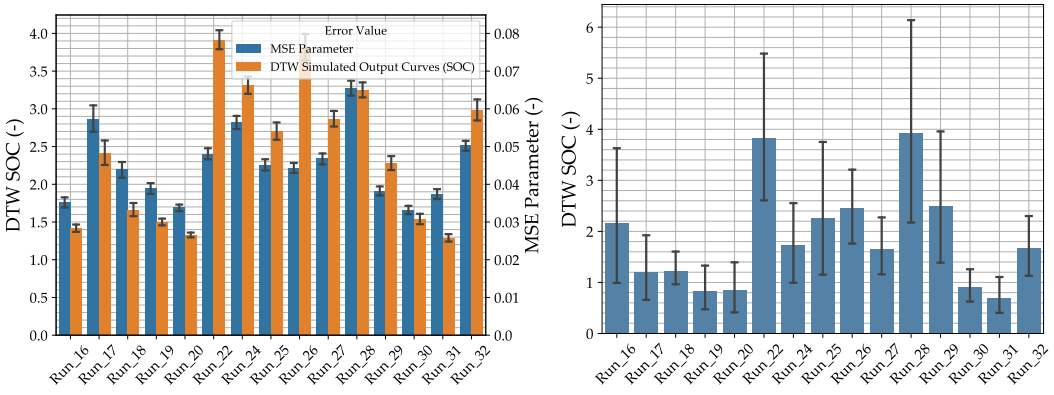
Figure 6 compares the distance measures of the five best runs of each network type with an additional 95% confidence interval, revealing the performance advantage of the CNN runs for both validation and experimental datasets. In Figure A1, the simulation results of run 19 are compared with those of the virtual experiment, whereby a good match can be observed. Except for run 21, which showed outlying high deviation in all error measures, the performances of all CNN runs are compared in Figure 7.



(**a**) (**b**)

Figure 6. Distance measure comparison for the five best runs of each network: (**a**) the validation dataset, and (**b**) the corresponding experimental dataset.

The higher performance regarding the prediction accuracy of the Default_CNN, compared to the remaining CNN types, is assumed to result from the relative simplicity of the network architecture (see Section 2.2.2). Unlike the other CNNs, this has no third convolutional block and no GAP or subsequent MLP layer, and possesses the smallest number of parameters due to the use of sparse connections and shared weights. Here, the CNN architecture seems to be of higher importance compared to the hyperparameters, since run 30 with HPO and the same architecture yielded comparable results. Partially, the other CNN types, such as CNN_Rautela, Resnet, and Encoder, achieved lower loss values for the training dataset compared to the Default_CNN, which suggests a too-high level of complexity of these CNN architectures regarding the present dataset. Thereby, the differences between the loss values for the training and the validation dataset are very large, which indicates overlearning. This can also be detected by observing the learning curves, whereas in Figure 8 the differences between training and validation loss of (a) Resnet and (b) Default_CNN are compared. The implemented early stopping prevented a further separation of the training and validation losses.



(**a**) (**b**)

Figure 7. Distance measure comparison for CNN runs on (**a**) the validation dataset and (**b**) the corresponding experimental dataset.

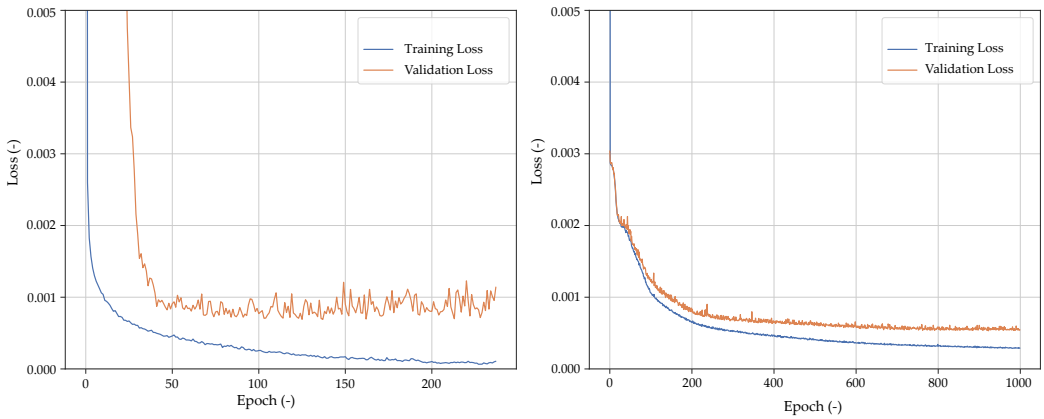


Figure 8. Learning curves of (**a**) Resnet run 25 and (**b**) Default_CNN run 19.

The worst performance in terms of DTW distance for validation and experimental dataset was obtained by run 21 with CNN_Yang and run 22 with CNN_Azizjon, these being networks using dropout, which could have a negative impact on the present dataset. The second-best performance of the CNNs from the literature regarding DTW distances was achieved by the encoder architecture.

To counteract overlearning and adapt the capacity of the networks to the fundamental time series regression problem, the number of parameters to be learned was reduced in FCN2 and FCN3 compared to FCN. This was done by significantly decreasing the number of filters in all three convolutional blocks. FCN2 omitted batch normalization. Based on the DTW distance of the validation dataset, it can be seen that this was successful. This is primarily valid for FCN3. The omission of the batch normalization conversely showed a negative effect on the prediction accuracy, which should be verified for other NN architectures in the future. Generally, the comparison between MLP and CNN architectures suggests a greater potential for increasing the prediction accuracy by CNNs. This may be due to the reduction in learnable network parameters of CNNs compared to MLPs due to the sparse connections and shared weights, and the consideration of local dependencies of data points.

In contrast, the BNNs based on the MLPs showed the lowest performance of the three investigated network variants. However, the reason may have been the complex and resource-intensive training process of the BNNs due to the large number of network parameters to be adjusted. Even after 20,000 epochs, the training and validation loss were still improving, which is why a continuation would probably lead to better results. Nevertheless, even for the small number of 20 to 30 hidden layer neurons, the training required several days, resulting from the high number of network parameters to be optimized. Since this number is also significantly dependent on the input point number of time series, this was reduced, whereas if the reduction is too large, as in run 35, this will lead to an SOC that is no longer representative and is therefore of limited applicability. In future investigations, further possibilities should be searched, e.g., by adapting the underlying network architecture. Due to the limitation of implementing only one subsequent stochastic layer, as described in Section 2.2.3, a larger number of neurons could be used in the first HL. This proved to be beneficial with respect to reducing the required epochs and training time and increasing the prediction accuracy; and again, a larger dataset led to better results. Varying the parameters of the normal distribution of the prior for runs 40 to 43 showed no advantage; thus, the common approach of a zero mean and a standard deviation of one is probably an appropriate starting point. Other distribution functions could be tested in future studies. As for the other network types, the use of multiple noise amounts further increased the prediction accuracy, but the training loss showed very high deviation from the corresponding validation loss.

Nevertheless, the main advantage of this network type is the distinction and quantification of the aleatoric and epistemic uncertainties. Figure 9 shows by example the effect of aleatoric uncertainties on the simulation with the predicted material parameters of run 39. This was obtained by sampling from the resulting probability density function (PDF), leading to a variety of material parameter sets, and each of these was used to simulate the ten experimental tests. Since the implementation of epistemic uncertainties leads to a divergent output for each prediction, 100 times the 500 MP sets of the PDF were sampled, and these 100 sets were then averaged. This result shows the underlying uncertainty of the data, which cannot be reduced by adding more datasets but is inherent to the data. Additionally, the experimental curve and the simulation result obtained from the point estimate which was calculated from the predicted PDF are shown. This displays that the underlying sensitivity of the MP on the simulation results has a varying impact on the different experiments. For example, the aleatoric uncertainties of the MP prediction have only a relatively small effect on the PEC of the compression test (Figure 9b) and the FDC of the bending test (Figure 9c). In contrast, they have a comparatively large impact on the FDC of the tensile test V3 (Figure 9a) and the FDC of the punch test (Figure 9d). Furthermore, it can be seen in Figure 9b that regardless of the uncertainties, the BNN was not able to predict the exact MP of $\nu_{p,press} = 0.5$, which would have led to the exact simulation result. The reason is probably that the searched material parameter also lay exactly on the MP range boundary.

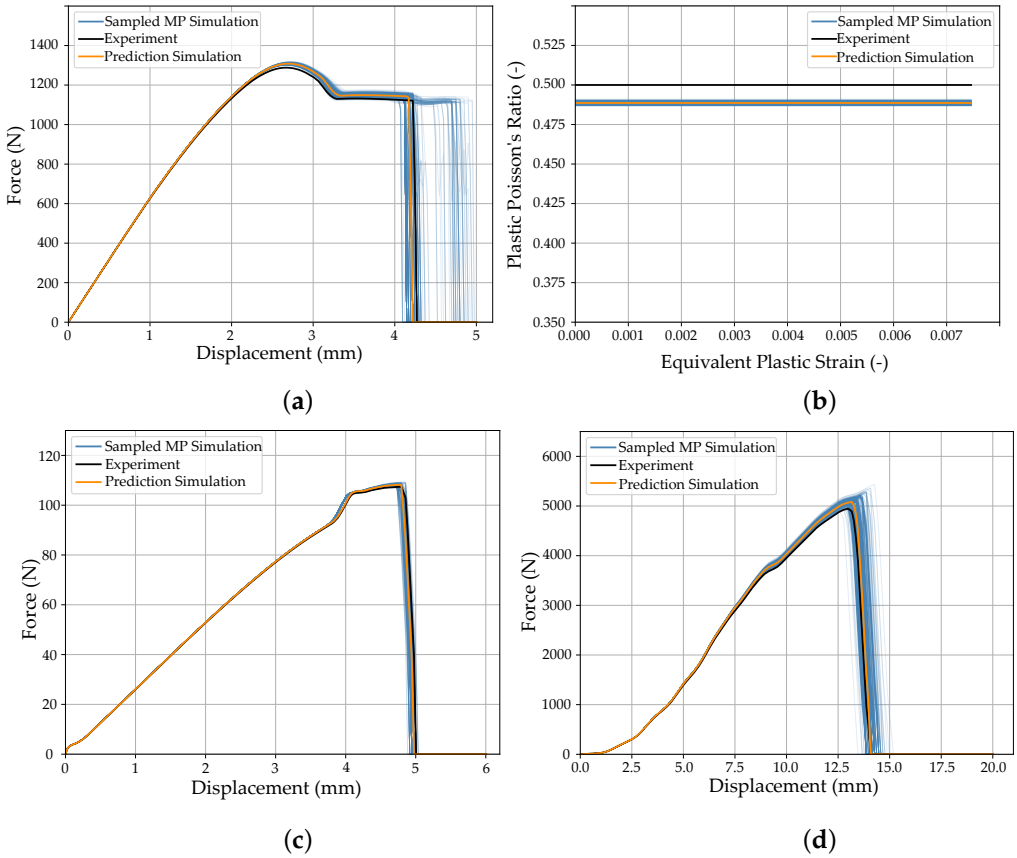
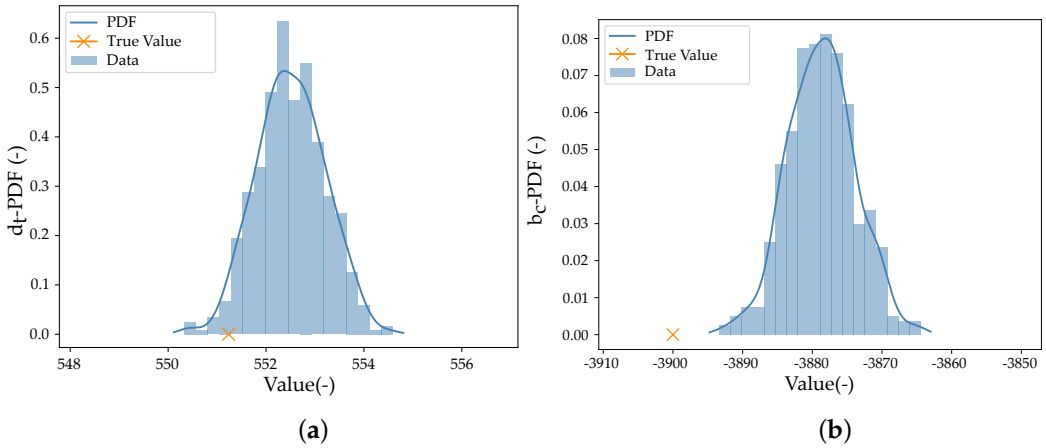


Figure 9. Comparison of the influence of aleatoric uncertainties on the experimental dataset simulation output for run 39: (**a**) FDC tension test V3, (**b**) PEC compression test, (**c**) FDC bending test, and (**d**) FDC punch test.

Figure 10 shows the probability density functions of two estimated material parameters. The true MP can also lie outside the PDF, which is not necessarily due to a poor prediction. Since the MP are coupled with each other and different MP combinations can

lead to similar or maybe even the same simulation results, there probably exist different MP combinations that can similarly reproduce the relationship to be learned. Therefore, the plot of the simulation results based on the sampled MP shown in Figure 9 provides additional value for assessing the reliability of the subsequent numerical simulations.

In the Appendix in Figure A2, the predicted MP with the true values and the aleatoric uncertainties (standard deviations) for the Runs 36 to 39 and 44 are shown. Despite the expectation of aleatoric uncertainties remaining constant for increasing dataset size, they decreased slightly (see also Table A10). Thus, the Gaussian noise amounts may not have been the main cause for the aleatoric uncertainties. Their origin lay instead in the data generation process and the occasionally unpredictable heavy oscillations, which could not be completely eliminated from the time series using the applied functions. Filter functions or other numerical effects during the simulation and the presence of many nonunique MP could also contribute to this.



(**a**) (**b**)

Figure 10. Probability density function of run 39 with the virtual experiment set of (**a**) MP d_t and (**b**) MP b_c.

In contrast, the more significant reducibility of epistemic uncertainties by using larger datasets can be seen in Figure 11. Normalized standard deviations of mean MPs and STDV of MPs for the validation and the experimental dataset are shown on the graphs. These resulted because a different probability density function was obtained for each prediction based on the weights and biases sampled for each prediction. This PDF can be described by the calculated statistics mean and STDV, whereas in the present case, the STDV of these means and STDVs can be calculated for the number of 100 performed predictions per BNN. For both validation and experimental datasets, a descending trend with a higher number of data can be observed for the mean and for the STDV. An exception is the MPs' STDV of the means of the second smallest dataset from the experimental dataset. This could be an indication that that model could produce a better predictions when repeating the stochastic training process with a constant number of data, and repetition could be advantageous. Based on the curve plot, it is reasonable to assume that further increasing the size of the dataset could lead to further reductions in epistemic uncertainties. However, in Table A10, when comparing the aleatoric and epistemic uncertainties, it can be seen that the latter are relatively small compared to the aleatoric uncertainties. Thus, the aleatoric uncertainties contributed more to the uncertainties of the prediction.

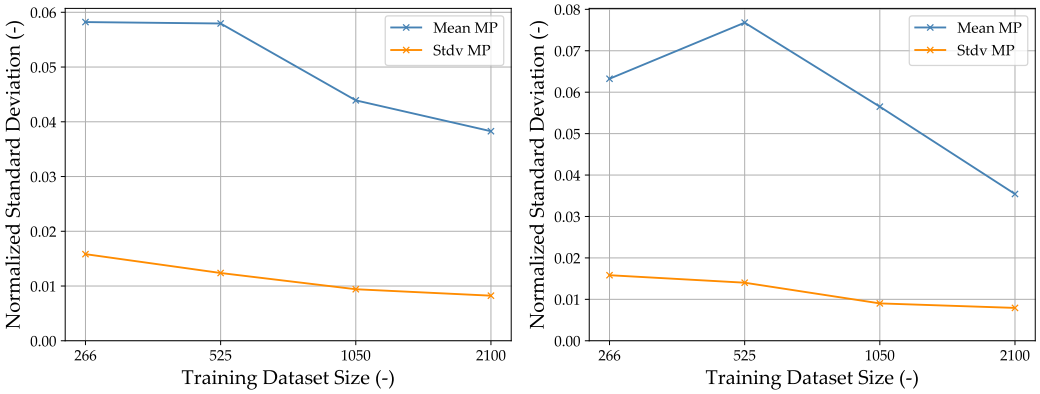


Figure 11. Epistemic uncertainties of different runs with varying dataset sizes of (**a**) validation dataset and (**b**) virtual experimental dataset.

4. Conclusions and Outlook

In this paper, the influences of different settings and architectures of MLP, CNN, and BNN on the prediction accuracies of material parameters were compared using the direct inverse MPI method in a comparative study. Thereby, 1D convolutional CNN network architectures proved to be especially capable of achieving high prediction accuracy, and in particular, the architecture of Zhao et al. [58] was able to achieve the best results for the present dataset. Most of the other CNN architectures from the literature tended to overlearn due to their high capacity. However, using a customized network, it was shown that by decreasing the capacity, e.g., by reducing filters, their prediction accuracy could be further increased, which suggests high potential for improvements. Furthermore, the application of batch normalization also positively affected prediction accuracy.

Furthermore, two data augmentation methods were investigated, whereby the use of different Gaussian noise amounts applied to the training data led to a further increase in the prediction accuracy and thus proved to be beneficial not exclusively due to the enhancement of the network's generalization capability. The application of the employed data decomposition method EMD did not improve any network's performance. However, this could have been due to the specific implementation and should be examined in further studies with various settings, in addition to other data augmentation methods.

The performances of the MLP-based BNNs did not reach those of the MLPs and CNNs, which might have been due to the resource-intensive training process caused by the large number of network parameters (distribution parameters and covariance matrix) and necessary complex computations. Hence, future work should investigate further possibilities, such as network complexity reduction, to further increase the prediction accuracy of BNNs. In conclusion, regarding prediction accuracy, CNNs are superior to at least the investigated MLP and BNN variants, potentially, among other things, due to their underlying grid structure and the consideration of the position of the data points. The sparse connections and shared weights lead to relatively short training durations. Although, if the network architecture is inappropriately chosen, overlearning can lead to significantly reduced prediction accuracies. However, the investigated BNNs demonstrated their advantage in terms of evaluability of prediction accuracy in the subsequent numerical simulation using the quantified aleatoric and epistemic uncertainties. Regarding the performance increase and reduction in the training duration, a test of the performances of BNNs based on CNNs would also be conceivable wherein the sparse connections could further reduce the number of network parameters. Additionally, the integration of the custom loss function or COL into the negative log-likelihood could be beneficial. Major advantages of the Bayesian network type are the assessable aleatoric and epistemic

uncertainties, which can be significant criteria for the evaluation of the simulation results of later component calculations.

Author Contributions: P.M. conceptualized the paper and designed the methodology of the digital experiments. P.M. developed the used Python code and conducted digital experiments and the analysis. T.H. provided input for the code and reviewed the paper. P.M. wrote the first draft, and T.V. supervised the research and contributed to the revision and editing of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge support by the Open Access Publication Funds of the Technische Universität Braunschweig.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ABS	Acrylonitrile Butadiene Styrene
AI	Artificial Intelligence
AUG	Augmentation
BNN	Bayesian Neural Network(s)
BCNN	Bayesian Convolution Neural Network(s)
CAE	Computer Aided Engineering
CNN	Convolutional Neural Network
CL	Custom Loss
CLF	Custom Loss Function
COL	Combined Loss Function
DA	Data Augmentation
DTW	Dynamic Time Warping
DVL	DenseVariational Layer
EMD	Empirical Mode Decomposition
EPS	Equivalent Plastic Strain
EPSF	Equivalent Plastic Strain at Failure
FCL	Fully Connected Layer
FCN	Fully Convolutional Neural Network
FDC	Force–Displacement Curve(s)
FE	Finite Element
FFANN	FeedForward Artificial Neural Network
GAP	Global Average Pooling
GD	Gradient Descent
GISSMO	Generalized Incremental Stress State-Dependent Damage Model
HL	Hidden Layer(s)
HP	Hyperparameter(s)
HPO	Hyperparameter Optimization
IMF	Intrinsic Mode Function
LHS	Latin Hypercube Sampling
MCDCNN	Multi-Channel Deep Convolutional Neural Network
MCIC	Material Card Input Curve(s)
ML	Machine Learning
MLP	Multilayer Perceptron
MP	Material Parameter(s)

MPI	Material Parameter Identification
MSE	Mean Squared Error
MTS	Multivariate Time Series
NLL	Negative Log-Likelihood
PDF	Probability Density Function
PEC	Plastic Poisson's Ratio – Equivalent Plastic Strain Curve(s)
PI	Parameter Identification
PPR	Plastic Poisson's Ratio
SANN	Stochastic Artificial Neural Networks
SOC	Simulation Output Curve(s)
STDV	Standard Deviation
TFP	TensorFlow Probability

Appendix A

Table A1. Non-default constants and settings of the virtual dataset. In the solver LS-DYNA, the inputs are specified without units. The unit system mm · t · s was used.

Input Name	Value
ro	1.04×10^{-9}
nue	0.35
dtyp	1.0
ecrit	0.0
dmgexp	1.0
dcrit	1.0

Table A2. Material parameters of the virtual experimental dataset and the corresponding MP ranges. Additionally, the corresponding material card input curve names are listed. In the Solver LS-DYNA, the inputs are specified without units. The unit system mm · t · s was used.

MP Name	MP_{exp}	MP_{Min}	MP_{Max}	MCIC
emod (MPa)	2127.3	2000.0	2300.0	-
a_t (-)	43,416.1	40,000.0	46,000.0	lcid-t; lcid-t1–lcid-t4
b_t (-)	−3934.39	−4100.00	−3700.00	lcid-t; lcid-t1–lcid-t4
c_t (-)	9.702	8.000	10.500	lcid-t; lcid-t1–lcid-t4
d_t (-)	551.24	540.00	562.00	lcid-t; lcid-t1–lcid-t4
a_c (-)	51,000.0	47,000.0	52,000.0	lcid-c
b_c (-)	−3900.00	−4100.00	−3800.00	lcid-c
c_c (-)	12.500	9.500	13.000	lcid-c
d_c (-)	550.00	540.00	560.00	lcid-c
$\nu_{p,plat}$ (-)	0.2265	0.1900	0.2900	lcid-p
$\nu_{p,press}$ (-)	0.5000	0.4000	0.5000	lcid-p
$\epsilon_{p,plat}$ (-)	0.1232	0.1000	0.1500	lcid-p
C ($\frac{1}{s}$)	27,572.1	15,000.0	50,000.0	lcid-t1–lcid-t4
P (-)	3.7482	2.7000	4.2000	lcid-t1–lcid-t4
$epsf_0$ (-)	0.3000	0.2500	0.4500	lcsdg
$epsf_1$ (-)	0.0500	0.0400	0.0550	lcsdg
$epsf_2$ (-)	0.0400	0.0340	0.0430	lcsdg
$epsf_3$ (-)	0.2400	0.2100	0.2800	lcsdg

Table A3. MLP architecture settings (* = default parameter).

(Hyper-)Parameter	Default_MLP	MLPHPO
Batch Size	25	40
Maximum Epochs *	500	500
Early Stopping Patience *	60	60
Neurons (IL) *	2400	2400
Hidden Layers	1	2
Neurons (HL1)	100	330
Kernel Initializer (HL1)	He Uniform	He Uniform
Activation (HL1)	Hard Sigmoid	Hard Sigmoid
Dropout (HL1)	0.10	0.05
Neurons (HL2)	-	400
Kernel Initializer (HL2)	-	Normal
Activation (HL2)	-	Softsign
Dropout (HL2)	-	0.05
Neurons (Output Layer) *	19	
Kernel Initializer (OL)	He Uniform	He Uniform
Activation (OL) *	Linear	Linear
Gradient Descent Optimizer	Adam	Adamax
HP Optimizer	-	Bayesian
HPO max. Trials	-	500

Table A4. MLP hyperparameter optimization search ranges (* = default parameter).

(Hyper-)Parameter	Search Range
Batch Size	20; 25 *; 30; . . . ; 150
Number HL	1 *; 2; 3
Neurons (HL1)	30; 40; 50 *; . . . ; 500
Kernel Initializer (HL)	Normal *; Uniform; Glorot Uniform; Lecun Uniform; Glorot Normal; He Normal; He Uniform
Activation (HL1)	Softmax; Softplus; Softsign; Relu *; Sigmoid; Hard Sigmoid
Dropout (HL1)	0.000; 0.025; 0.050 *; . . . ; 0.250
Neurons (HL2)	30; 40; 50 *; . . . ; 500
Kernel Initializer (HL2)	Normal *; Uniform; Glorot Uniform; Lecun Uniform; Glorot Normal; He Normal; He Uniform
Activation (HL2)	Softmax; Softplus; Softsign; Relu *; Sigmoid; Hard Sigmoid
Dropout (HL2)	0.000; 0.025; 0.050 *; . . . ; 0.250
Neurons (HL3)	30; 40; 50*; . . . ; 500
Kernel Initializer (HL3)	Normal *; Uniform; Glorot Uniform; Lecun Uniform; Glorot Normal; He Normal; He Uniform
Activation (HL3)	Softmax; Softplus; Softsign; Relu *; Sigmoid; Hard Sigmoid
Dropout (HL3)	0.000; 0.025; 0.050 *; . . . ; 0.250
Kernel Initializer (OL)	Normal *; Uniform; Lecun Uniform; Glorot Normal; He Normal; He Uniform
GD Optimizer	Adam *; Adagrad; Adamax; Nadam

Table A5. CNN architecture settings.

(Hyper-)Parameter	Default_CNN	CNNHPO	FCN2	FCN3
Batch Size	16	16	16	16
Maximum Epochs	1000	1000	1000	2000
Early Stopping Patience	100	100	none	100
Layer 1	Conv1D	Conv1D	Conv1D	Conv1D
Layer 1 Filters	6	12	12	12
Layer 1 Kernel Size	7	2	8	8
Layer 1 Activation	Sigmoid	Tanh	Relu	Relu
Layer 1 Batch Normalization	no	no	no	yes
Layer 1 Pooling	Average Pooling1D	Average Pooling1D	no	no
Layer 1 Pooling Size	3	7	-	-
Layer 2	Conv1D	Conv1D	Conv1D	Conv1D
Layer 2 Filters	12	128	24	24
Layer 2 Kernel Size	7	2	5	5
Layer 2 Activation	Sigmoid	Tanh	Relu	Relu
Layer 2 Batch Normalization	no	no	no	yes
Layer 2 Pooling	Average Pooling1D	Average Pooling1D	no	no
Layer 2 Pooling Size	3	5	-	-
Layer 3	-	-	Conv1D	Conv1D
Layer 3 Filters	-	-	24	24
Layer 3 Kernel Size	-	-	3	3
Layer 3 Activation	-	-	Relu	Relu
Layer 3 Batch Normalization	-	-	no	yes
Layer 3 Pooling	-	-	no	no
Layer 4	Flatten	Flatten	Global Average Pooling1D	Global Average Pooling1D
Dense Output Layer Neurons	18	18	18	18
Gradient Descent Optimizer	Adam	Adam	Adam	Adam
HP Optimizer	-	Bayesian	-	-
HPO max. Trials	-	250	-	-

Table A6. CNN hyperparameter optimization search range.

(Hyper-)Parameter	Search Range
Layer 1 Filters	3; 6; 12; 16; 32; 64; 128
Layer 1 Kernel Size	2; 3; 5; 7; 9
Layer 1 Pooling Size	2; 3; 5; 7
Layer 2 Filters	3; 6; 12; 16; 32; 64; 128
Layer 2 Kernel Size	2; 3; 5; 7; 9
Layer 2 Pooling Size	2; 3; 5; 7
Activation Function	Relu; Sigmoid; Tanh
GD Optimizer	Adam; Adamax; Nadam

Table A7. BNN architecture settings (building order).

(Hyper-)Parameter	BNN1	BNN2	BNN3
Batch Size	25	25	25
Maximum Epochs	20,000	20,000	5000
Early Stopping	none	none	none
Batch-Normalization-Layer	yes	yes	yes
Not Stochastic-Hidden FCL	no	no	yes
Not Stochastic-FCL Neurons	-	-	100
Not Stochastic-FCL Activation	-	-	Sigmoid
Not Stochastic-FCL Kernel-Initializer	-	-	Glorot Uniform
DVL Neurons	30	20	25
DVL Activation	Sigmoid	Sigmoid	Sigmoid
TFP Prior	Multivariate-NormalDiag	Multivariate-NormalDiag	Multivariate-NormalDiag
TFP Posterior	Multivariate-NormalTriL	Multivariate-NormalTriL	Multivariate-NormalTriL
kl_use_exact	True	True	True
TFP Output Layer	IndependentNormal	IndependentNormal	IndependentNormal
GD Optimizer	RMSprop	RMSprop	RMSprop

Table A8. Settings of the NN-based direct inverse MPI runs.

Run	NN	Dataset	Loss	Noise AUG	EMD AUG	Eval. Point Number SOC	Prior Mean	Prior SDV
1	Default_MLP	1	MSE	No	No	200	-	-
2	Default_MLP	2	MSE	No	No	200	-	-
3	Default_MLP	3	MSE	No	No	200	-	-
4	Default_MLP	4	MSE	No	No	200	-	-
5	Default_MLP	1	COL	No	No	200	-	-
6	Default_MLP	2	COL	No	No	200	-	-
7	Default_MLP	3	COL	No	No	200	-	-
8	Default_MLP	4	COL	No	No	200	-	-
9	Default_MLP	5	COL	No	No	200	-	-
10	Default_MLP	6	COL	No	No	200	-	-
11	Default_MLP	7	COL	No	No	200	-	-
12	Default_MLP	8	COL	No	No	200	-	-
13	MLPHPO	3	COL	No	No	200	-	-
14	Default_MLP	3	COL	Yes	No	200	-	-
15	Default_MLP	3	COL	No	Yes	200	-	-
16	Default_CNN	3	MSE	No	No	200	-	-
17	Default_CNN	1	COL	No	No	200	-	-
18	Default_CNN	2	COL	No	No	200	-	-
19	Default_CNN	3	COL	No	No	200	-	-
20	Default_CNN	4	COL	No	No	200	-	-
21	CNN_Yang	3	COL	No	No	200	-	-
22	CNN_Azizjon	3	COL	No	No	200	-	-
23	CNN_Rautela	3	COL	No	No	200	-	-
24	MCDCNN	3	COL	No	No	200	-	-

Table A8. *Cont.*

Run	NN	Dataset	Loss	Noise AUG	EMD AUG	Eval. Point Number SOC	Prior Mean	Prior SDV
25	Resnet	3	COL	No	No	200	-	-
26	FCN	3	COL	No	No	200	-	-
27	Encoder	3	COL	No	No	200	-	-
28	FCN2	3	COL	No	No	200	-	-
29	FCN3	3	COL	No	No	200	-	-
30	CNNHPO	3	COL	No	No	200	-	-
31	Default_CNN	3	COL	Yes	No	200	-	-
32	Default_CNN	3	COL	No	Yes	200	-	-
33	BNN1	3	NLL	No	No	50	0	1
34	BNN2	3	NLL	No	No	50	0	1
35	BNN1	3	NLL	No	No	30	0	1
36	BNN3	1	NLL	No	No	200	0	1
37	BNN3	2	NLL	No	No	200	0	1
38	BNN3	3	NLL	No	No	200	0	1
39	BNN3	4	NLL	No	No	200	0	1
40	BNN3	3	NLL	No	No	200	0	0.2
41	BNN3	3	NLL	No	No	200	0	2
42	BNN3	3	NLL	No	No	200	2	1
43	BNN3	3	NLL	No	No	200	0.25	1
44	BNN3	3	NLL	Yes	No	200	0	1
45	BNN3	3	NLL	No	Yes	200	0	1

Table A9. Additional Gaussian noise amounts (normalized by standard deviation) for noise data augmentation. The mean for each dataset is zero.

SOC Type	SDV1	SDV2	SDV3	SDV4
FOC	0.001	0.002	0.003	0.004
PEC	0.0005	0.0006	0.0007	0.0008

Table A10. Aleatoric and epistemic uncertainties of normalized MP for different BNN runs for experimental dataset. (The mean was calculated for all 18 MP).

Normalized Uncertainty (-)	Uncertainty Type	Run36	Run37	Run38	Run39	Run44
STDV	aleatoric	0.24221	0.21959	0.20172	0.20719	0.19264
STDV of Mean	epistemic	0.06325	0.07678	0.05650	0.03543	0.04427
STDV of STDV	epistemic	0.01582	0.01402	0.00902	0.00793	0.00797

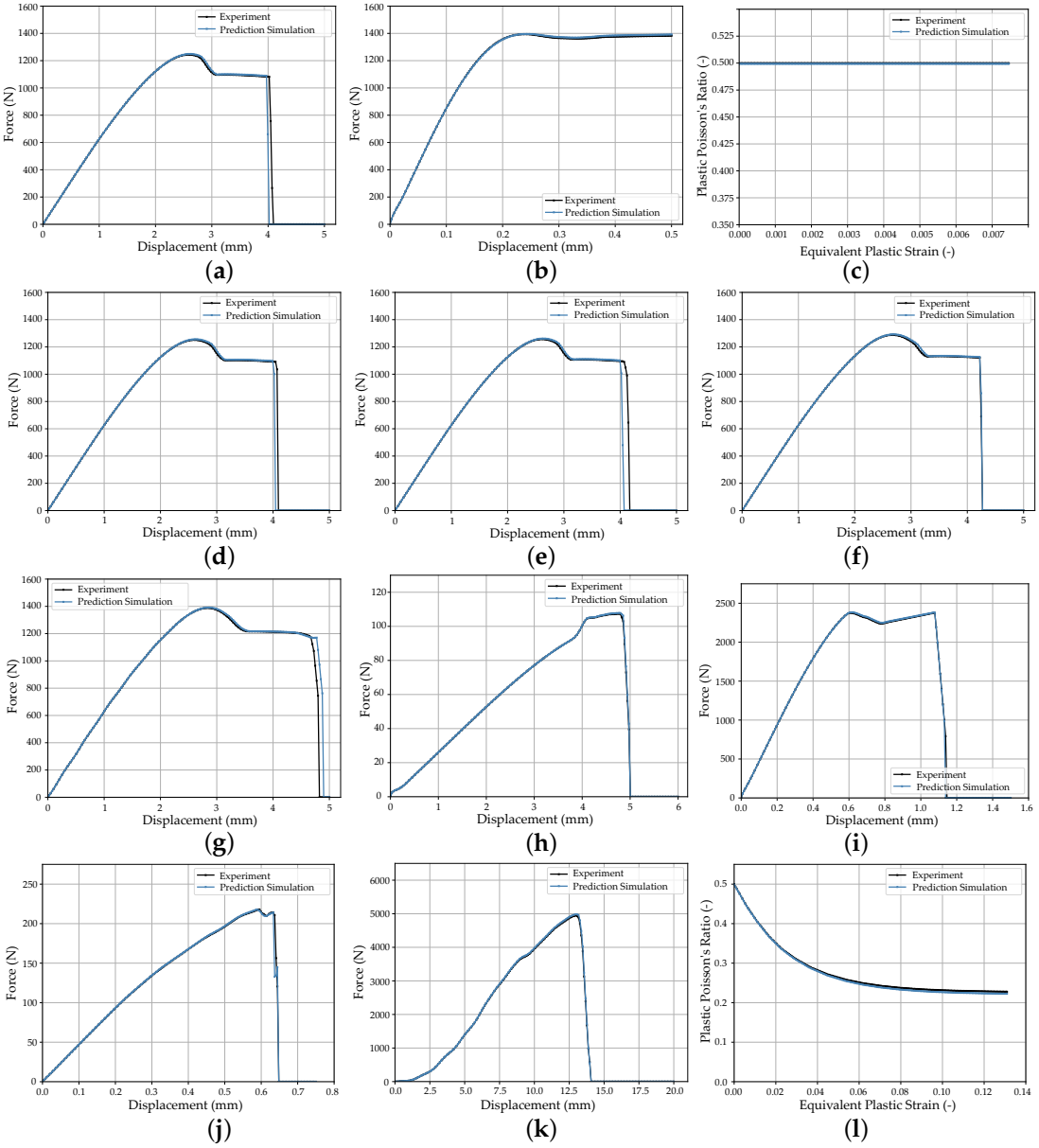


Figure A1. Comparison simulation output curves of run 19 with the virtual experimental test dataset: (**a**) FDC tension test, (**b**) FDC compression test, (**c**) PEC compression test, (**d**) FDC tension test V1, (**e**) FDC tension test V2, (**f**) FDC tension test V3, (**g**) FDC tension test V4, (**h**) FDC bending test, (**i**) FDC shear1 test, (**j**) FDC shear2 test, (**k**) FDC punch test, and (**l**) PEC punch test.

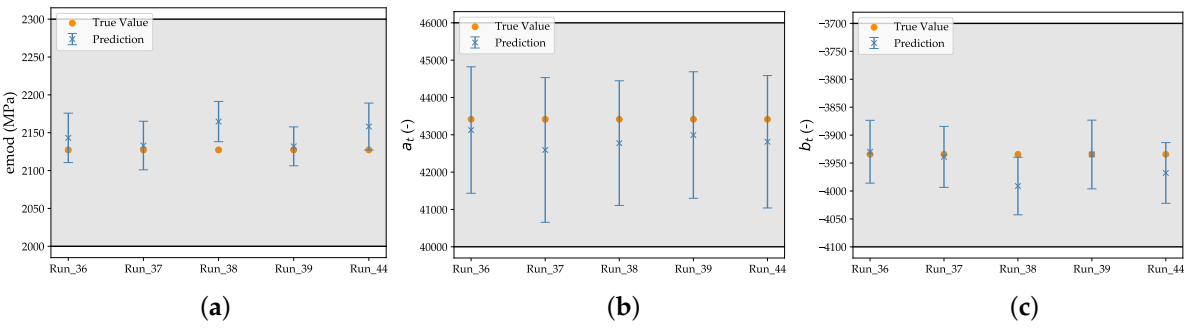


Figure A2. *Cont.*

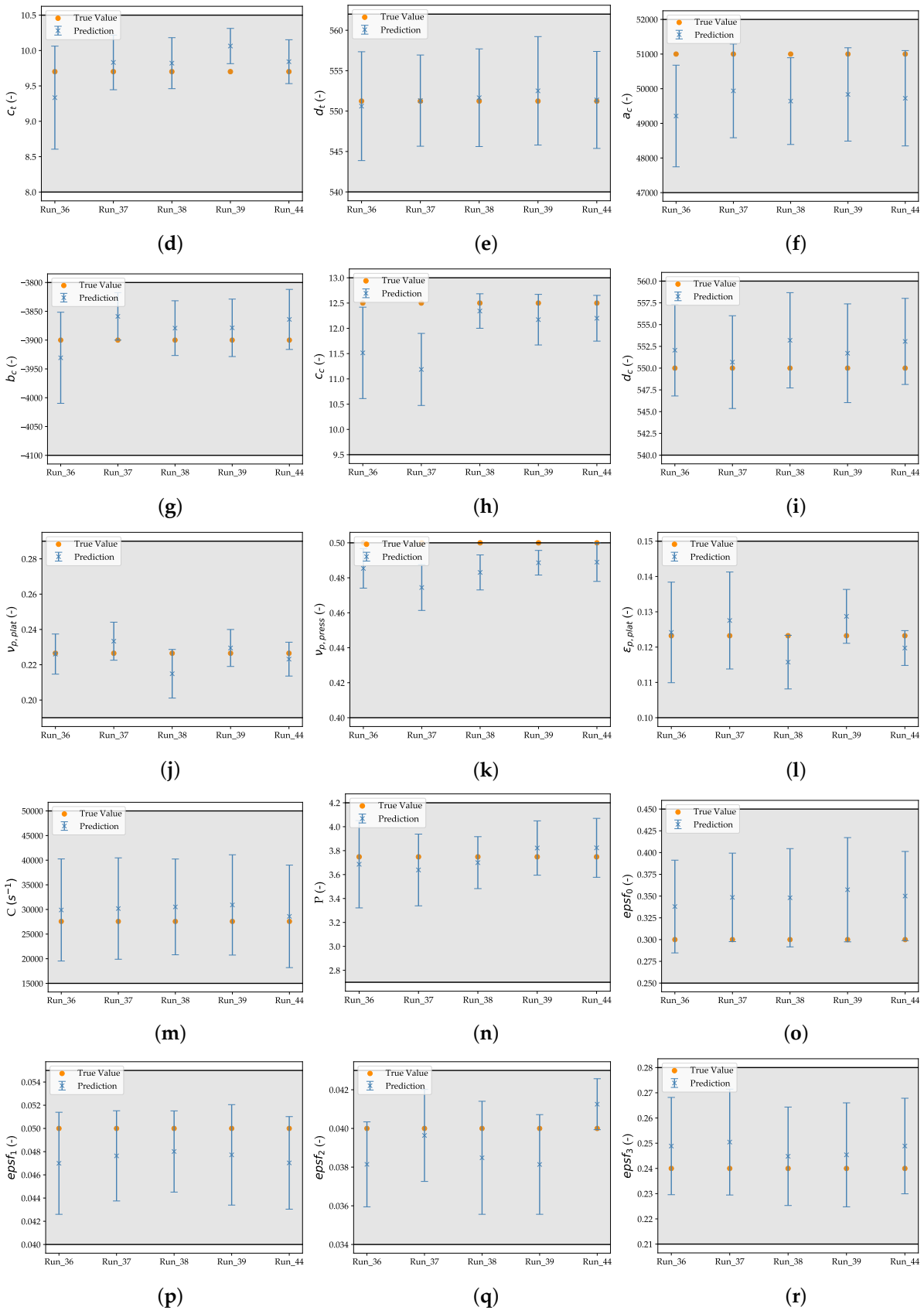


Figure A2. Deviation plots for different runs and all material parameters (**a**–**r**).

References

1. Kohar, C.P.; Greve, L.; Eller, T.K.; Connolly, D.S.; Inal, K. A machine learning framework for accelerating the design process using CAE simulations: An application to finite element analysis in structural crashworthiness. *Comput. Methods Appl. Mech. Eng.* **2021**, *385*, 114008. [CrossRef]
2. Jones, E.; Carroll, J.; Karlson, K.; Kramer, S.; Lehoucq, R.; Reu, P.; Turner, D. Parameter covariance and non-uniqueness in material model calibration using the Virtual Fields Method. *Comput. Mater. Sci.* **2018**, *152*, 268–290. [CrossRef]
3. Mahnken, R. Identification of Material Parameters for Constitutive Equations. In *Encyclopedia of Computational Mechanics Second Edition*; John Wiley & Sons: Hoboken, NJ, USA, 2018. [CrossRef]
4. Grabski, J.K.; Mrozek, A. Identification of elastoplastic properties of rods from torsion test using meshless methods and a metaheuristic. *Comput. Math. Appl.* **2021**, *92*, 149–158. [CrossRef]
5. Kolodziej, J.A.; Jankowska, M.A.; Mierzwiczak, M. Meshless methods for the inverse problem related to the determination of elastoplastic properties from the torsional experiment. *Int. J. Solids Struct.* **2013**, *50*, 4217–4225. [CrossRef]
6. Asaadi, E.; Wilke, D.N.; Heyns, P.S.; Kok, S. The use of direct inverse maps to solve material identification problems: Pitfalls and solutions. *Struct. Multidiscip. Optim.* **2016**, *55*, 613–632. [CrossRef]
7. Meißner, P.; Winter, J.; Vietor, T. Methodology for Neural Network-Based Material Card Calibration Using LS-DYNA MAT_187_SAMP-1 Considering Failure with GISSMO. *Materials* **2022**, *15*, 643. [CrossRef]
8. Morand, L.; Helm, D. A mixture of experts approach to handle ambiguities in parameter identification problems in material modeling. *Comput. Mater. Sci.* **2019**, *167*, 85–91. [CrossRef]
9. Yagawa, G.; Okuda, H. Neural networks in computational mechanics. *Arch. Comput. Methods Eng.* **1996**, *3*, 435–512. [CrossRef]
10. Abendroth, M.; Kuna, M. Determination of deformation and failure properties of ductile materials by means of the small punch test and neural networks. *Comput. Mater. Sci.* **2003**, *28*, 633–644. [CrossRef]
11. Chamekh, A.; Salah, H.B.H.; Hambli, R. Inverse technique identification of material parameters using finite element and neural network computation. *Int. J. Adv. Manuf. Technol.* **2008**, *44*, 173–179. [CrossRef]
12. Ktari, Z.; Leitão, C.; Prates, P.A.; Khalfallah, A. Mechanical design of ring tensile specimen via surrogate modelling for inverse material parameter identification. *Mech. Mater.* **2021**, *153*, 103673. [CrossRef]
13. Rappel, H.; Beex, L.A.A.; Hale, J.S.; Noels, L.; Bordas, S.P.A. A Tutorial on Bayesian Inference to Identify Material Parameters in Solid Mechanics. *Arch. Comput. Methods Eng.* **2019**, *27*, 361–385. [CrossRef]
14. Shridhar, K.; Laumann, F.; Liwicki, M. A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference. *arXiv* **2019**, arXiv:1901.02731. [CrossRef]
15. Gundersen, K.; Alendal, G.; Oleynik, A.; Blaser, N. Binary Time Series Classification with Bayesian Convolutional Neural Networks When Monitoring for Marine Gas Discharges. *Algorithms* **2020**, *13*, 145. [CrossRef]
16. MacKay, D.J.C. A Practical Bayesian Framework for Backpropagation Networks. *Neural Comput.* **1992**, *4*, 448–472. [CrossRef]
17. Jospin, L.V.; Laga, H.; Boussaid, F.; Buntine, W.; Bennamoun, M. Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Comput. Intell. Mag.* **2022**, *17*, 29–48. [CrossRef]
18. Unger, J.F.; Könke, C. An inverse parameter identification procedure assessing the quality of the estimates using Bayesian neural networks. *Appl. Soft Comput.* **2011**, *11*, 3357–3367. [CrossRef]
19. Bagnall, A.; Lines, J.; Bostrom, A.; Large, J.; Keogh, E. The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.* **2016**, *31*, 606–660. [CrossRef]
20. Kiranyaz, S.; Avci, O.; Abdeljaber, O.; Ince, T.; Gabbouj, M.; Inman, D.J. 1D convolutional neural networks and applications: A survey. *Mech. Syst. Signal Process.* **2021**, *151*, 107398. [CrossRef]
21. Munir, M.; Siddiqui, S.A.; Dengel, A.; Ahmed, S. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access* **2019**, *7*, 1991–2005. [CrossRef]
22. Demir, S.; Mincev, K.; Kok, K.; Paterakis, N.G. Data augmentation for time series regression: Applying transformations, autoencoders and adversarial networks to electricity price forecasting. *Appl. Energy* **2021**, *304*, 117695. [CrossRef]
23. Liu, B.; Zhang, Z.; Cui, R. Efficient Time Series Augmentation Methods. In Proceedings of the 2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP—BMEI), Chengdu, China, 17–19 October 2020; IEEE: Piscataway, NJ, USA, 2020, pp. 1004-1009. [CrossRef]
24. Iwana, B.K.; Uchida, S. An empirical survey of data augmentation for time series classification with neural networks. *PLoS ONE* **2021**, *16*, e0254841. [CrossRef] [PubMed]
25. Winkler, P.; Koch, N.; Hornig, A.; Gerritzen, J. OmniOpt – A Tool for Hyperparameter Optimization on HPC. In *Lecture Notes in Computer Science*; Springer International Publishing: Berlin/Heidelberg, Germany, 2021; pp. 285–296.
26. Meißner, P.; Watschke, H.; Winter, J.; Vietor, T. Artificial Neural Networks-Based Material Parameter Identification for Numerical Simulations of Additively Manufactured Parts by Material Extrusion. *Polymers* **2020**, *12*, 2949. [CrossRef] [PubMed]
27. Kolling, S.; Haufe, A.; Feucht, M.; Bois, P.A.D. SAMP-1: A Semi-Analytical Model for the Simulation of Polymers. In Proceedings of the 4th LS-DYNA Anwenderforum, Bamberg, Germany, 20–21 October 2005.
28. Livermore Software Technology Corporation (LSTC). *LS-DYNA Keyword User's Manual Volume II Material Models LS-DYNA*, 11th ed.; LSTC: Livermore, CA, USA, 2018.
29. Andrade, F.X.C.; Feucht, M.; Haufe, A.; Neukamm, F. An incremental stress state dependent damage model for ductile failure prediction. *Int. J. Fract.* **2016**, *200*, 127–150. [CrossRef]

30. Hallquist, J.O. *LS-DYNA Theory Manual*; Livermore Software Technology Corporation (LSTC): Livermore, CA, USA, 2006.
31. Neukamm, F.; Feucht, M.; Haufe, A.D. Considering damage history in crashworthiness simulations. In Proceedings of the 7th European LS-DYNA Conference, Salzburg, Austria, 14–15 May 2009.
32. Basaran, M.; Wölkerling, S.D.; Feucht, M.; Neukamm, F.; Weichert, D. An Extension of the GISSMO Damage Model Based on Lode Angle Dependence. In Proceedings of the 9th LS-DYNA FORUM 2010, Stuttgart, Germany, 12–13 October 2010; DYNAmore: Stuttgart, Germany, 2010; pp. 3–17 .
33. Haufe, A.; DuBois, P.; Neukamm, F.; Feucht, M. GISSMO – Material Modeling with a sophisticated Failure Criteria. In Proceedings of the Conference LS-DYNA Info Day, Filderstadt, Germany, 13 October 2011. [CrossRef]
34. Lemaitre, J. A Continuous Damage Mechanics Model for Ductile Fracture. *J. Eng. Mater. Technol.* **1985**, *107*, 83–89. [CrossRef]
35. Lemaitre, J. *A Course on Damage Mechanics*; Springer: Berlin/Heidelberg, Germany, 1996. [CrossRef]
36. Škrlec, A.; Klemenc, J. Estimating the Strain-Rate-Dependent Parameters of the Cowper-Symonds and Johnson-Cook Material Models using Taguchi Arrays. *Strojniški Vestn. J. Mech. Eng.* **2016**, *62*, 220–230. [CrossRef]
37. Hayashi, S. Prediction of Failure Behavior in Polymers Under Multiaxial Stress State. *Seikei-Kakou* **2013**, *25*, 476–482. [CrossRef]
38. Stavroulakis, G.; Bolzon, G.; Waszczyszyn, Z.; Ziemianski, L. Inverse Analysis. In *Comprehensive Structural Integrity*; Elsevier: Amsterdam, The Netherlands, 2003; pp. 685–718. [CrossRef]
39. Goldberg, D.E. Genetic algorithms in search, optimization, and machine learning. *Comput. Sci.* **1989**, *27*, 27-0936. [CrossRef]
40. Stander, N.; Craig, K.; Müllerschön, H.; Reichert, R. Material identification in structural optimization using response surfaces. *Struct. Multidiscip. Optim.* **2005**, *29*, 93–102. [CrossRef]
41. Kučerová, A. Identification of Nonlinear Mechanical Model Parameters Based on Softcomputing Methods. Ph.D. Thesis, Czech Technical University, Prague, Czech Republic, 2007.
42. Aguir, H.; BelHadjSalah, H.; Hambli, R. Parameter identification of an elasto-plastic behaviour using artificial neural networks–genetic algorithm method. *Mater. Des.* **2011**, *32*, 48–53. [CrossRef]
43. Wen, Q.; Sun, L.; Yang, F.; Song, X.; Gao, J.; Wang, X.; Xu, H. Time Series Data Augmentation for Deep Learning: A Survey. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, Montreal, Canada, 19–27 August 2021; pp. 4653–4660. [CrossRef]
44. Shorten, C.; Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* **2019**, *6*, 1–48. [CrossRef]
45. Huang, N.E.; Shen, Z.; Long, S.R.; Wu, M.C.; Shih, H.H.; Zheng, Q.; Yen, N.C.; Tung, C.C.; Liu, H.H. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proc. R. Soc. London Ser. A Math. Phys. Eng. Sci.* **1998**, *454*, 903–995. [CrossRef]
46. Stander, K.W.N. Modified Dynamic Time Warping for Utilizing Partial Curve Data to Calibrate Material Models. In Proceedings of the 16th International LS-DYNA Users Conference, Virtual Event, 31 May–2 June 2020.
47. Petitjean, F.; Ketterlin, A.; Gançarski, P. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognit.* **2011**, *44*, 678–693. [CrossRef]
48. Giorgino, T. Computing and Visualizing Dynamic Time Warping Alignments inR: ThedtwPackage. *J. Stat. Softw.* **2009**, *31*, 1–24. [CrossRef]
49. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 23 June 2022).
50. Haykin, S. Number Bd. 10. In *Neural Networks and Learning Machines*; Prentice Hall: Hoboken, NJ, USA, 2009.
51. da Silva, I.N.; Spatti, D.H.; Flauzino, R.A.; Liboni, L.H.B.; dos Reis Alves, S.F. *Artificial Neural Networks*; Springer International Publishing: Berlin/Heidelberg, Germany, 2017. [CrossRef]
52. Fausett, L.; Fausett, L. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*; Prentice-Hall: Hoboken, NJ, USA, 1994.
53. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; IEEE: Piscataway, NJ, USA, 2015. [CrossRef]
54. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
55. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv* **2014**, arXiv:1409.0473. [CrossRef]
56. Gamboa, J.C.B. Deep Learning for Time-Series Analysis. *arXiv* **2017**, arXiv:1701.01887. [CrossRef]
57. Fawaz, H.I.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.* **2019**, *33*, 917–963. [CrossRef]
58. Zhao, B.; Lu, H.; Chen, S.; Liu, J.; Wu, D. Convolutional neural networks for time series classification. *J. Syst. Eng. Electron.* **2017**, *28*, 162–169. [CrossRef]
59. Wang, Z.; Yan, W.; Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14-19 May 2017; IEEE: Piscataway, NJ, USA, 2017. [CrossRef]
60. Yang, H.; Meng, C.; Wang, C. Data-Driven Feature Extraction for Analog Circuit Fault Diagnosis Using 1-D Convolutional Neural Network. *IEEE Access* **2020**, *8*, 18305–18315. [CrossRef]

61. Azizjon, M.; Jumabek, A.; Kim, W. 1D CNN based network intrusion detection with normalization on imbalanced data. In Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 19–21 February 2020; IEEE: Piscataway, NJ, USA, 2020. [CrossRef]
62. Rautela, M.; Gopalakrishnan, S. Deep Learning frameworks for wave propagation-based damage detection in 1D-waveguides. In Proceedings of the 11th International Symposium on NDT in Aerospace, Paris, France, 13–15 November 2020.
63. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Front. Comput. Sci.* **2015**, *10*, 96–112. [CrossRef]
64. Serrà, J.; Pascual, S.; Karatzoglou, A. Towards a universal neural network encoder for time series. *arXiv* **2018**, arXiv:1805.03908. [CrossRef]
65. Abdar, M.; Pourpanah, F.; Hussain, S.; Rezazadegan, D.; Liu, L.; Ghavamzadeh, M.; Fieguth, P.; Cao, X.; Khosravi, A.; Acharya, U.R.; et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Inf. Fusion* **2021**, *76*, 243–297. [CrossRef]