







Article

Deep Learning as a New Framework for Passive Vehicle Safety Design Using Finite Elements Models Data

Mar Lahoz Navarro ¹, Jonas Siegfried Jehle ², Patricia A. Apellániz ¹, Juan Parras ^{1,*}, Santiago Zazo ¹
and Matthias Gerdtz ³

¹ Information Processing and Telecommunications Center, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain

² Passive Safety Department, Bayerische Motoren Werke Group, 80809 Munich, Germany

³ Department of Aerospace Engineering, University of the Bundeswehr Munich, 85579 Neubiberg, Germany

* Correspondence: j.parras@upm.es

Abstract: In recent years, passive vehicle safety has become one of the major concerns for the automotive industry due to the considerable increase in the use of cars as a means of daily transport. Since real crash testing has a high financial cost, finite element simulations are generally used, which entail high computational cost and long simulation times. In this paper, we make use of the recent advances in the deep learning field to propose an affordable method to provide reliable approximations of the finite element simulator model that significantly reduce the computational load and time required. We compare the prediction performance in crash tests of different models, namely feed-forward neural networks and bayesian neural networks, as well as two multi-output regression methods. Our results show promising results, as deep learning models are able to drastically reduce the engineering costs while providing a feasible first approximation to the passenger's injuries in a crash event, thus being a potential game changer in the vehicle safety design process.

Keywords: passive vehicle safety; crashworthiness; finite elements; feed-forward neural network; bayesian neural network



Citation: Lahoz Navarro, M.; Jehle, J.S.; Apellániz, P.A.; Parras, J.; Zazo, S.; Gerdtz, M. Deep Learning as a New Framework for Passive Vehicle Safety Design Using Finite Elements Models Data. *Appl. Sci.* **2024**, *14*, 9296. <https://doi.org/10.3390/app14209296>

Academic Editors: Domenico Mazzeo and Aleš Dittrich

Received: 12 September 2024

Revised: 4 October 2024

Accepted: 10 October 2024

Published: 12 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, the use of cars as a means of daily transport has considerably increased. Even so, despite this increase in car use, in recent decades the number of accidents has fallen substantially, going from 54,000 road fatalities in 2001 in the European Union to 25,100 in 2018 [1]. This is largely due to the effort car companies invest in making vehicles safer. However, the decrease in the number of accidents has stagnated the last few years [2], having become the most common cause of death and injuries. In Europe, the most recent data show 20,400 road fatalities in 2023; thus, the progression in road fatality reduction is stalling [3]. Therefore, more changes need to be made in order to achieve the challenging target of halving the 2020 road accidents and people seriously injured in the European Union by 2030 proposed in [1].

In order to reach the previous ambitious goal by 2030, a deeper understanding of the factors influencing an accident must be gathered, and prediction and prevention of traffic accidents should be further investigated. The recent advances in the machine learning field are being used to provide insights on many of the factors that contribute to the road fatalities, such as using machine learning to identify the key variables that have a significant contribution to fatal road injuries [4], to assess the pavement condition using image segmentation techniques [5], or, especially, to predict the severity of a crash [6–11].

Nevertheless, one of the most influential factors in a traffic accident and one of the factors that can have the greatest influence on minimizing passenger injuries is the design of the passive vehicle safety features, e.g., restraint systems, which are the ones that we focus on in this work. According to [12], passive vehicle safety elements come into force

in the event of a collision, helping to minimize the impact received by the passenger and posterior injuries. Within the passive safety field, the term “crashworthiness” can be found repeatedly in existing research [13]: in the automotive engineering context, this term is related to the capacity of the vehicle structure to protect its occupants in the event of collision. In order to actually assess whether a certain vehicle complies with safety standards and regulations, and before a new car model is introduced on the market, it must be subjected to vehicle safety assessment programs, e.g., in Europe, these are conducted by the European New Car Assessment Program (Euro NCAP). These assessment programs require the use of Anthropomorphic Test Devices (ATDs), also called crash test dummies, to determine the capacity of the car to protect its occupants in a crash event. To achieve that target, test dummies are equipped with a multitude of sensors capable of precisely capturing different measurements of interest exerted on each body part.

However, real car crash testing demands high financial costs. To reduce the number of hardware crash tests and to better understand the underlying system, Finite Element (FE) simulations are widely used, e.g., to reconstruct the collision of a motorcycle and a car [14], to study the response of a car to a frontal crash [15], and to simulate crash scenarios [16]. Thus, FE is the state-of-the-art technique in the passive safety design process, as a recent review on crashworthiness shows [13]. FE tools lower monetary costs and speed up the process to some extent because they can simulate the real world with little error [13], but at the same time, they entail high computational costs, implying long computational times. Thus, in order to obtain the measures of interest in the test dummy during a crash event (i.e., the neck acceleration during a frontal crash), we need to run a FE model, which is usually solved by a commercial software that acts as a black box, providing only inputs and outputs of the model. This FE model takes a long time to simulate, consuming a significant amount of resources. For instance, the data used in the simulations of this paper come from a FE simulator that takes over 45 h to compute each simulation on 36 CPUs. Due to these reasons, approximation techniques, also known as meta-models, are applied to build approximate mathematical models of complex models that provide a rough but fast approximation to the crash curves [17]. In this work, we propose taking advantage of the recent advances in deep learning (DL), so we use the approximation capabilities of neural networks (NNs) in order to estimate the crash curves.

In several fields where FE simulations are used, machine learning tools are being applied due to their advantages. In [18], the authors review several applications in the biomedical field, such as modeling tissue deformation, and report significant gains in simulation time compared to FE methods. In [19], the authors show that brain injury can be predicted using DL tools and report significant time gains compared to FE simulations. Another field where these ideas are applied is in the civil engineering field to predict the behavior of complex structures, such as a dam in a seismic scenario [20] or the structural level strength of composite laminates [21]. However, in vehicle safety, we are only aware of the use of DL techniques to evaluate the crash response of a mechanical bump absorber [22], but to the best of our knowledge, we are the first ones to propose the use of DL to predict the crash severity on a test dummy. Our results show that the significant savings in time and computational resources make our proposed procedure a potential game changer in the passive safety design process, as DL can be used to approximate crash curves in order to accelerate the design process compared with relying solely on FE methods.

Hence, the main contributions of this paper are as follows:

- We propose several architectures based on DL to evaluate the severity of a crash, taking as input the parameters of the restraint system. Namely, we propose using feed-forward NNs (FFNNs) and bayesian NNs (BNNs).
- We also discuss the important topic of uncertainty by making use of BNNs, which not only provide a predicted output but also a measure of the uncertainty of that output that can be used to identify situations where minimal changes do have a strong impact on the crash severity and focus the design in such situations.

- In order to assess the performance of the proposed models, we use real data from FE simulations in the BMW Passive Safety Department. Due to the computational load required for each of these simulations, we only have 140 crash test sequences to validate our DL models, which, however, show very promising results, both in terms of good prediction capability and a significant reduction in computational load. FE models require 36 CPUs and more than 45 h of processing time, but DL models train in minutes using a GPU.
- In order to facilitate the replicability and foster future research, we publish the data from the FE simulations used in this paper along with the code needed to replicate our results, so that future research can use our code and/or data as a baseline.

The rest of the paper goes as follows: in Section 2 we present the input and output data used in this work; in Section 3 we explain the theoretical foundations of the algorithms used; and then in Section 4 we present the most relevant results from our research; and finally, in Section 5 we draw the conclusions that stem from this work.

2. Material

In order to find the parametric function that best approximates crash responses, the data used in this paper comes from a Full Frontal Crash model and is generated using a commercial FE simulator. The data obtained using this FE simulator is $\{x_i, y_i\}_{i=1}^N$, where $i \in \{1, 2, \dots, N\}$ indexes the number of input–output vectors generated, $x \in \mathbb{R}^4$ are the input parameters, and $y \in \mathbb{R}^M$ represent the output curve. Hence, note that our problem consists of predicting y given x , which is a multi-output regression problem due to the fact that y is a vector of dimension M , where M is the number of time steps simulated. Due to the computational requirements of FE models, we generate $N = 140$ pairs of input–output data, which take 45 h of processing using 36 CPUs. Let us delve into more detail on the meaning of x and y .

2.1. Input Data

Each input data vector x_i consists of four parameters that give information on the settings of the restraint system, more precisely, on (1) settings of the passenger airbag, (2) the knee airbag, (3) the belt configuration, and (4) the airbag time-to-fire. We sampled them using a uniform distribution with values normalized between 0 and 1, as seen in Figure 1. Note that these parameters do not depend on time.

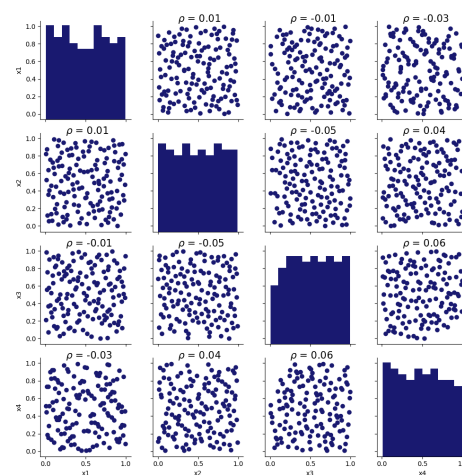


Figure 1. Correlation matrix plot for the four input parameters (passenger and knee airbag, belt configuration and airbag time-to-fire). We obtain $N = 140$ crash responses using a uniform distribution over these four parameters.

2.2. Output Data

Each output data y_i consists of a discrete time curve, i.e., y_i is an M -dimensional vector, with $M = 1150$ in our case. We use y_i to refer to the i output curve and y_i^j to refer to the j time step of the i curve, where $i \in \{1, 2, \dots, N\}$ and $j \in \{1, 2, \dots, M\}$. Each curve y_i represents some measurements of interest of the dummy in the car crash event, which are the neck y-moment of the passenger and the resultant head acceleration. We select these two measurements due to their significance for determining passenger injuries. An overview of the shape of each of these curves can be seen in Figure 2, where we can observe that these curves have a very non-linear shape, with one or two peaks of different amplitudes in different time steps, which means that the prediction problem is not immediate.

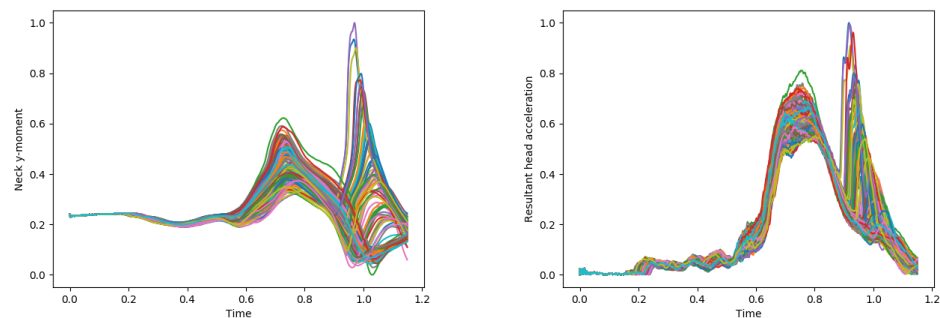


Figure 2. Representation of output crash responses for the neck y-moment (left) and the head acceleration (right) for each of the $N = 140$ curves present in our data. The horizontal axis represents the time, where we note that each curve is composed of $M = 1150$ time steps. Note that the first curves may or may not have a second peak starting at $t \approx 0.9$, while the second may present a second peak of varying amplitude at $t \approx 1$, which indicates a complex behavior.

3. Theoretical Background

Since our aim is to find the parametric functions that best approximate crash responses, we will carry out a comparison between selected multi-output regression methods. We devote this section to introducing the basic theory needed to understand each of these methods.

3.1. Baseline Models

In order to benchmark the results of the DL methods that we propose, we select two methods to compare with, namely, the Stacked Single-Target (SST) method and a modified Ensemble of Regression Chains (ERC), both of them being state-of-the-art methods in multi-output regression [23,24]. It is important to note that these two methods build an independent regression model for each time step in the output curve y .

3.1.1. Stacked Single-Target

An SST regressor consists of training a Single-Target regressor for each element of the output curve [23], which means that each Single-Target regressor takes as input x_i and estimates y_i^j . Thus, we need M regressors that share the same input x and have as targets a different time step of the output curve y . Under this approach, the multi-output regression problem is broken down into single-output regression problems, and thus, any regression methods could be applied. In mathematical terms, we have that each of the M Single-Target regressors solves the following problem:

$$\hat{y}_i^j = f(x_i), \quad j \in \{1, 2, \dots, M\} \quad (1)$$

where f represents the chosen regressor. The total output curve is estimated by a global model created by stacking each of the individual predictions: $\hat{y}_i = [\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^M]$. Due to the significant computational effort that is required to train $M = 1150$ regressors, we

choose to implement each Single-Target regressor as an independent Linear Regressor to predict each of the time steps of the output curve.

3.1.2. Ensemble of Regression Chains

This method is similar to the previous one in that it builds a classifier for each time step of the output, but the difference now is that the inputs differ for each of these classifiers. Namely, this method considers as input to each regressor, in addition to the input parameters x , some previous time steps of the output curve [24]. Although we could consider all previous time steps for predicting, we implement a modified scheme in which we consider a window of a limited number of previous targets instead of all the previous targets. Moreover, for each target, the best model is selected in terms of some evaluation metrics (R-squared and Mean Squared Error (MSE)) out of different combinations of the four input parameters and the previously predicted targets within the proposed window. In mathematical terms, we have that each of the M regressors solves the following problem:

$$\hat{y}_i^j = f\left(x_i, y_i^{j-L}, y_i^{j-L+1}, \dots, y_i^{j-1}\right), \quad j \in \{1, 2, \dots, M\} \quad (2)$$

where f represents the chosen regressor and L is the window size used. Again, the total output curve is estimated by a global model created by stacking each of the individual predictions: $\hat{y}_i = [\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^M]$.

As mentioned, both SST and ERC build an independent regression model for each time step in the output curve y , but we know that there is a significant correlation between predicted outputs, as can be seen in Figure 2. While the SST method does not account for this correlation, the ERC does by taking as an additional input a window of previous outputs.

3.2. Deep Learning Models

Due to the proven ability of DL models to find complex relationships between input–output pairs in a multitude of different problems by creating hierarchical interconnected layers consisting of many units, we consider two different architectures to address the multi-output regression problem at hand. We now proceed to introduce the two DL architectures chosen, based on FFNNs and BNNs, and also explain why another popular architectural choice, Recurrent NNs (RNNs), does not provide an advantage to our problem.

3.2.1. Feed-Forward Neural Networks

FFNNs, also known as Multi-Layer Perceptrons (MLPs), are the most basic DL architectures. FFNNs are able to model complex relationships between input–output pairs by composing together many functions [25] and natively support multi-output regression problems. Moreover, as the universal function approximation theorem states [26], under mild conditions, they are capable of finding the function that relates input to output with a certain level of precision.

FFNNs consist of interconnected layers composed of different numbers of neurons, in which the information is progressively broken down into features and flows in one direction. In Figure 3, we show a schematic diagram of a three-layered FFNN for illustration purposes. On it, the following are differentiated: the input layer, which is responsible for receiving the input vector and passing it to the next layer; the hidden layer, which performs all the computations; and finally, the output layer, which generates the predicted values. Each layer consists of many units, called neurons, that are fully connected with the immediately adjacent layers in one direction; note that we may add many hidden layers, which tends to increase the prediction performance when enough data are available [25].

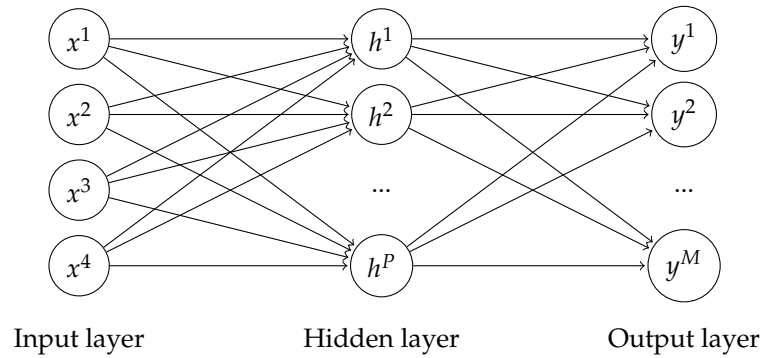


Figure 3. Three-layered FFNN for our problem, where each circle represents a neuron, and neurons are grouped in layers; for easier view, we have drawn a single hidden layer of dimension P . Note that each neuron takes as input all the neurons of the previous layer and computes a single output using the expression (3).

The fundamental computational unit of a neural network is referred to as a neuron, or unit. As described in [27], a neuron is composed of three main elements: the connecting links, weighted by a numerical value known as weight, w_{ji} , which indicates the relevance of such connection in the output value; the summing junction, which sums all the incoming input values; and the activation function, $\varphi(\cdot)$, which limits the dynamic range of the output value, which depends itself on the chosen function. Some of the most applied activation functions are the linear function, the Sigmoid, the Hyperbolic tangent or the Rectified Linear Unit. In addition, there is a scalar value known as bias, b_i . Moreover, each neuron has a single output and receives many inputs. Thus, the output is a non-linear combination of all the inputs, and the equation that computes the output value ζ_i^j of neuron i in layer j can be described as

$$\zeta_i^j = \varphi \left(\sum_{k=1}^P w_{k,i} \zeta_k^{j-1} + b_i \right) \tag{3}$$

where ζ_k^{j-1} are the input values to layer j neurons, i.e., the output of the P neurons of the previous layer $j - 1$, and φ represents the activation function chosen.

The most popular method to train FFNNs is the Back-Propagation (BP) algorithm. The training proceeds in two phases [27]: the forward and the backward propagation. In the former, weights and biases are randomly initialized, and input values are passed through the layers performing operation 3 on each neuron until the output layer is reached. Then, the error between the predicted value and the actual value is computed. This error is propagated backwards towards the input layer, and the weights and bias values of each neuron are adjusted until the error converges to an acceptable level. This is referred to as the backward propagation and is an optimization problem in which a certain loss function has to be minimized by adjusting the weights and biases of the NN. As a loss function, we use the Mean Squared Error (MSE), which is widely used in regression problems and is described by the following equation:

$$\text{MSE} = \frac{1}{NM} \sum_{i=1}^N \sum_{j=1}^M (y_i^j - \hat{y}_i^j)^2 \tag{4}$$

where y_i^j represents the true values and \hat{y}_i^j are the predictions of the network for trajectory i and time step j . Detailed computations of the Back-Propagation algorithm are presented in [25,27]. The optimization algorithm that we use is Adam [28]. It is a first-order method based on gradient, whose name is derived from the estimation of the adaptive momentum.

3.2.2. Bayesian Neural Networks

It is important to note that FFNNs are deterministic: given a certain input, they always provide the same output. They also do not account for the uncertainty of the prediction, i.e., they do not provide any measurement of their certainty of the given prediction, which is known as calibration in the literature [29]. In order to address both issues, we use BNNs, whose main difference to FFNNs is that the weights and biases, instead of being a scalar, are now considered a Gaussian random variable with mean and variance. Thus, the training differs, and it is now based on an algorithm called Bayes By Backpropagation [30], based on the Bayes' Theorem.

The fact that the weights and biases in a BNN are random variables means that each time that we provide an input to a BNN, the weights are sampled according to their distribution, and thus, the output will differ even if the input is the same. This means that we can predict several outputs for a fixed input, and by studying the distribution of the outputs, we can assess whether the BNN is confident in its prediction (i.e., low variance) or not. That has been shown to improve the calibration of the network [29], and it is useful because it allows identifying the outputs where the BNN has a lower certainty, which is important to our problem, as we will see in Section 4. Moreover, it allows controlling the prediction error, which is something that must be controlled even in the case of using FE [13].

3.2.3. Recurrent Neural Networks

Finally, and for completeness, we also discuss RNNs in this part of our study, as these are specialized in treating time sequences. The main difference between a FFNN and a RNN lies in the fact that an RNN has a feedback loop, as seen in Figure 4. Mathematically, RNNs are specialized in processing time series information: for each input element of the input sequence, x_t , it produces an output y_t and a hidden state h_t that contains information that is then introduced as an additional input for x_{t+1} . This means that there are two actual inputs: x_t and h_{t-1} , and thanks to the information contained in h , the RNN is able to propagate information through time. They are trained with the Back-Propagation Through Time (BPTT) algorithm [31], which allows obtaining a meaningful value of the hidden state so that it feedbacks past decisions to have relevance for future ones. There are many possible RNN architectures, being Long-Short Term Memory (LSTM) [32] one of the most popular.

Although RNNs are widely applied in sequential problems, in our case, they do not provide better results than FFNNs. This is due to a subtle fact that is related to how RNNs are used to feedback information through the hidden state h : in general, when RNNs are applied, the output sequence is conditioned on previous information that impacts the next outputs. Let us illustrate this with the text captioning problem, as shown in [33–35], where an image is processed and the caption text is automatically generated using an LSTM. Given the image features x , we want to generate a sequence of words y that captions the text. Since the output y is the probability of choosing a certain word, note that it does matter whether the first word y_1 of our text is "A" or "He," as in the former case, the next word must be a noun or an adjective, whereas in the latter it should be a verb. So note that y_2 , the second word in the caption, depends on the sampled word y_1 : the hidden state h_1 contains that information, which is new and needed to predict y_2 . Hence, to compute the next work, we not only need the previous distribution over words, but the actual word sampled.

However, in our problem, which, recalling it once again, consists of finding the function that best relates multi-dimensional input parameters to the corresponding output response, this sampling of random variables is not present. But, in contrast, the hidden state that would be present in the RNN network would not add any new information that was not already present at the outset, since the hidden state would be a deterministic transformation of the input parameters x . Hence, the hidden state is nothing more than a more complex transformation of the input values due to the architecture of RNN, but it

does not add any new information. That is why there is no benefit in applying RNNs to address the proposed problem in this paper, so we do not report their results in Section 4, because they are not better than those provided by FFNNs.

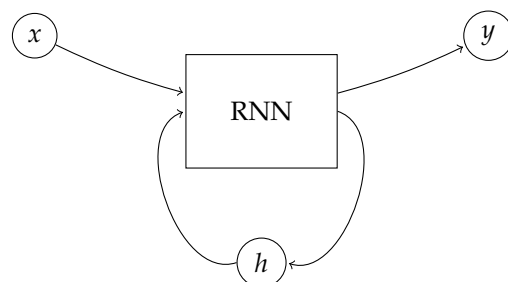


Figure 4. Illustration of the structure of an RNN, where we emphasize that, apart from the input x and the output y , we have a hidden state h . For each time step t , the network takes as input x_t and the previous hidden state, h_{t-1} , and produces two outputs, y_t and h_t . Note that the RNN is able to propagate information through time thanks to the hidden state h_t .

4. Results

We are now ready to show the performance results of the proposed methods in our problem, which we do in this section. First, in Section 4.1, we compare FFNNs to the proposed baseline methods; then, in Section 4.2, we compare the performance of FFNNs and BNNs; and finally, in Section 4.3, we discuss the results obtained.

In order to assess the predictive performance, we use three different metrics widely used in regression problems:

- Mean Squared Error, which has already been introduced in (4). IT is widely used as it heavily penalizes large deviations in the predicted values from the ground truth, but the units of the MSE are not the ones of the target since they are squared. It goes between 0 and $+\infty$, and the lower, the better.
- Mean Absolute Error (MAE), which is computed as the average absolute error between the ground truth and the predicted values. It gives an idea of the error in the units of the target. It goes between 0 and $+\infty$, and the lower, the better.
- The R^2 metric, also called coefficient of determination, determines how well the model fits the data (goodness of fit). It goes between 0 and 1, and the higher, the better.

4.1. Baselines vs. Feed-Forward Neural Network

We start by comparing our baseline methods to the FFNN. In order to find the most optimized architecture for the FFNN in terms of the number of layers, the number of neurons per layer, and the activation per layer, we have used Hyperas [36] to find the best candidate parameter out of 2000 possible combinations. In order to preprocess the input data for the FFNN, we normalize the values of the data between 0 and 1.

The baseline methods are the SST method introduced in Section 3.1.1 and the ERC presented in Section 3.1.2. Let us remind that for SST, each regressor takes as input the four input parameters, whereas for ERC, we consider as input different combinations of the four input parameters and the immediately previous time steps, which increases the input dimensionality. To keep it bounded, we only test combinations using four previous time steps.

Finally, regarding the ERC method, we note that during the training phase, real values of previous time steps are considered as input, while during the prediction phase, we introduce as input the previously predicted values. It is important to note that we are trying to propose an efficient model to replace FE simulations due to their long computing times, and although the methods proposed in here are not even comparable in terms of required time, these methods are computationally inefficient and require more time both in the training and prediction stages, while providing worse results than FFNNs, as seen in the next sections.

4.1.1. Neck y-Moment

We first present results corresponding to the neck y-moment. This particular curve is interesting because of the two different behaviors that are present in $t \approx 0.9$ that can be observed in Figure 2. Models are expected to identify whether the direction of the second peak goes upwards or downwards. The training parameters are depicted in Table 1. Moreover, the optimized architecture that has been found with Hyperas is in Table 2.

Table 1. Selected training parameters for FFNN.

Parameter	Selected Value
Learning rate	10^{-3}
Optimizer	Adam
Objective function	MSE
Number of epochs	350
Batch size	100, all training samples
Validation split	0.1

Table 2. Selected hyperparameters for FFNN in the neck y-moment.

Parameter	Selected Value
Number of layers	5
Neurons per layer	(1024, 512, 64, 1024, 1150)
Activation per layer	(ReLU, ReLU, Sigmoid, ReLU, Linear)

Once the model has been trained, its performance is assessed with the validation set. Two predicted curves for the three different models are presented in Figure 5. It can be observed that the two proposed multi-output regression models are incapable of accurately predicting the behavior of the second peak, while the FFNN does. This is also reflected in the evaluation metrics presented in Table 3, where it can be observed that FFNNs offer the best performance, while SST and ERC provide similar results, worse than the FFNN, as expected.

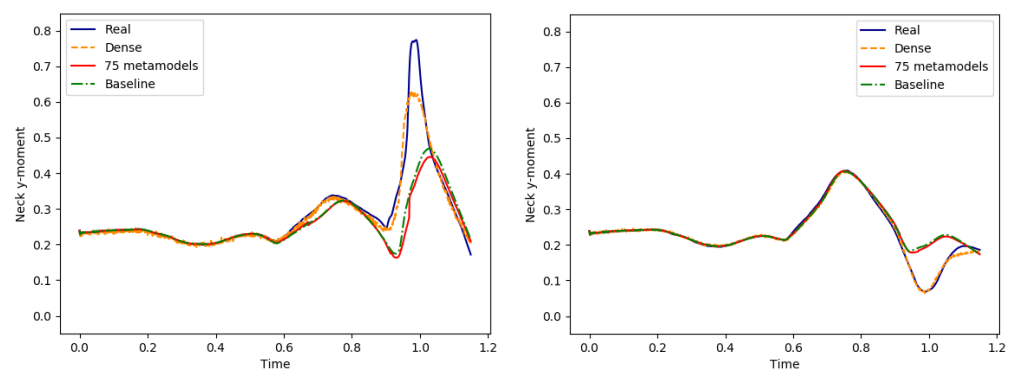
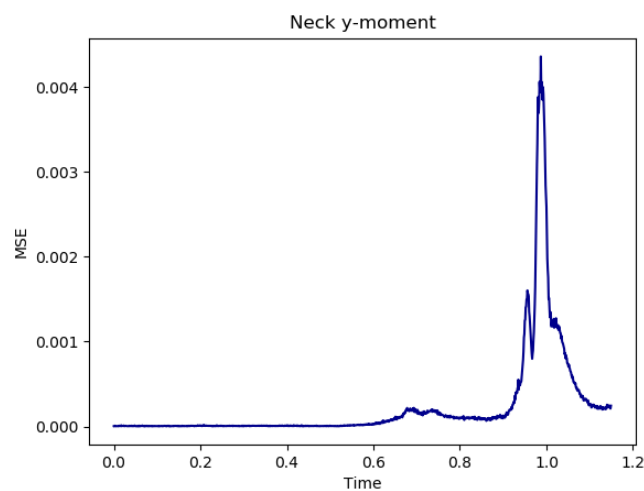


Figure 5. Comparison between predicted curves for the three proposed methods and the real curve chosen for two different test samples for the neck y-moment. “Real” stands for the ground truth, “Dense” for the FFNN prediction, “Baseline” for SST prediction, and “75 metamodels” for ERC prediction. Note that the best prediction is clearly given by the FFNN method, which is the one that best predicts the second peak in the first curve.

Table 3. Evaluation metrics comparison between SST, ERC, and FFNNs for the neck y-moment output curve.

Metric	SST	ERC	FFNN
R^2 (higher is better)	0.842	0.850	0.965
MSE (lower is better)	$1.258 \cdot 10^{-3}$	$1.203 \cdot 10^{-3}$	$2.331 \cdot 10^{-4}$
MAE (lower is better)	$1.435 \cdot 10^{-2}$	$1.382 \cdot 10^{-2}$	$6.669 \cdot 10^{-3}$

Finally, Figure 6 represents the average MSE distribution for all testing samples along time, in which it can be observed that around $t = 1$, the error is greater, which corresponds to the second peak seen in Figure 2. Note that this is to be expected: this is the most complex region of the signal to predict, and FFNNs accumulate their error here. However, as noted before, the network does not output a measure of the uncertainty in this region, as BNNs do.

**Figure 6.** Average MSE as a function of time for the neck y-moment output response. Note that the error peaks in the part where the predicted signal becomes more complex, as seen in Figure 2.

4.1.2. Resultant Head Acceleration

We now show the results obtained in the resultant head acceleration prediction. As the previous output response was selected due to the two different behaviors it has around $t = 1$, this one is selected due to the relevance it has in determining the passenger injuries: it predicts whether an impact of the head with the dashboard has occurred. The training parameters are the same as the ones used before and are depicted in Table 1. In addition, the optimized architecture can be found in Table 4.

Table 4. Selected hyperparameters for FFNN in the resultant head acceleration.

Parameter	Selected Value
Number of layers	5
Neurons per layer	(512, 512, 64, 1024, 1150)
Activation per layer	(ReLU, Tanh, Sigmoid, ReLU, Linear)

After training the model, new output curves are predicted with the testing set, and two predicting examples are represented in Figure 7. These two curves correspond to the same two test samples presented in Figure 5, so both figures have been predicted with the same set of input parameters. The evaluation metrics are presented in Table 5, where it is shown that FFNNs, once again, performed better than the baseline methods. Moreover, as it can be observed in the predicted curves depicted in the first part of Figure 7, the FFNN is not able to perfectly fit the second peak of the curve. However, it is able to determine

that a large-scale second peak has occurred, unlike the other two methods. Thus, it can be concluded that the FFNN is capable of detecting whether an impact of the head with the dashboard has happened.

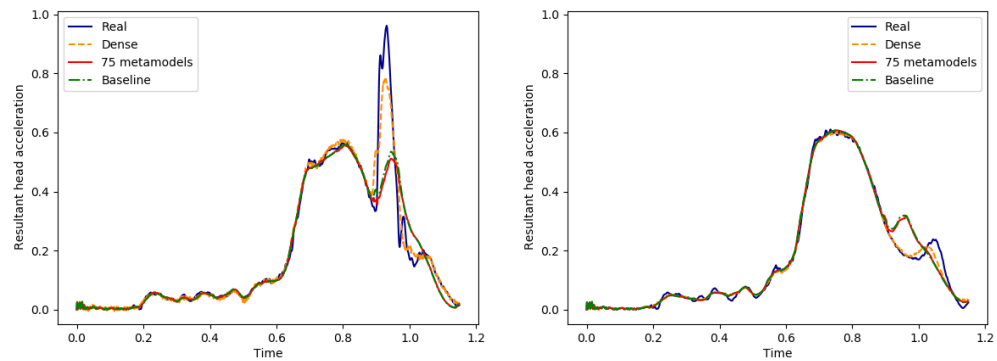


Figure 7. Comparison between predicted curves for the three proposed methods and the real curve chosen for two different test samples for the head acceleration. “Real” stands for the ground truth, “Dense” for the FFNN prediction, “Baseline” for SST prediction, and “75 metamodels” for ERC prediction. Note that the best prediction is clearly given by the FFNN method, which is able to predict the occurrence of a large peak in the first curve, while the other methods fail to predict it.

Table 5. Evaluation metrics comparison between SST, ERC, and FFNNs for the resulting head acceleration output curve.

Metric	SST	ERC	FFNN
R^2 (higher is better)	0.976	0.977	0.993
MSE (lower is better)	$1.043 \cdot 10^{-3}$	$1.013 \cdot 10^{-3}$	$3.153 \cdot 10^{-4}$
MAE (lower is better)	$1.340 \cdot 10^{-2}$	$1.309 \cdot 10^{-2}$	$8.543 \cdot 10^{-3}$

Finally, the mean MSE value along time for all test samples is presented in Figure 8, where it can be observed that, for this output response, the MSE error is concentrated before reaching $t = 1$, which again corresponds to the most complex region of the data, as observed in Figure 2.

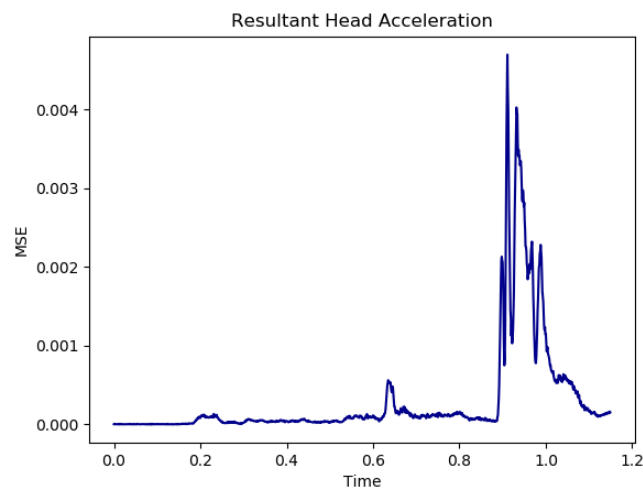


Figure 8. Average MSE as a function of time for the head acceleration output response. Again, note that the error peaks in the parts of the curve where the behavior becomes more complex to predict.

4.2. Feed-Forward Neural Network vs. Bayesian Neural Network

As our aim is to find a reliable model to predict output responses for some new input parameters, we would like to determine how reliable the predictions given by the models are, and to that purpose, we use BNNs. By applying these networks, we will know in which parts of the output curves the model is more doubtful and to what extent by having confidence intervals. We use the same hyperparameters (number of layers, neurons per layer, and activations) that we found for FFNNs (Tables 2 and 4), and again, we normalize the data between 0 and 1.

In BNNs, since weights follow a probability distribution, different predictions are obtained for the same set of input parameters. This means that we can obtain a prediction average and confidence interval, which in this work is obtained using the 3σ rule of the normal distribution, in which the 99.73% of values lie. This phenomenon can be assumed due to the central limit theorem, which states that, having n independent random variables drawn for any distribution with known mean μ and variance σ^2 greater than 0 and finite, the normalized sum of these independent random variables can be characterized by a normal distribution:

$$Z = \lim_{n \rightarrow \infty} \left(\frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \right) \rightarrow N(0, 1) \tag{5}$$

being \bar{X} the sample mean and n the number of samples. In our case, we will test using $n = 100$ different draws of the output for each input in order to obtain a good estimate of the confidence intervals.

In order to present the results, once again, we divide this section into two parts: the first one will present the results obtained for the neck y -moment, and the second part will present the ones obtained for the resultant head acceleration.

4.2.1. Neck y -Moment

We train both an FFNN and a BNN using the architecture from Table 2, using the training parameters presented in Table 6. The parameters considered are based on the ones used in [37], which are, in turn, based on the ones presented in [30].

Table 6. Selected training parameters for BNNs.

Parameter	Selected Value
Learning rate	10^{-4}
Optimizer	Adam
Objective function	MSE
Number of epochs	150
Batch size	100, all training samples
Validation split	0.1
μ_{weight}	Normal (0, 0.1)
μ_{bias}	Normal (0, 0.1)
ρ_{weight}	Normal (−5, 0.05)
ρ_{bias}	Normal (−5, 0.05)
ϵ	Normal (0, 0.1)
σ_1	e^{-1}
σ_2	e^{-6}
Scale Mixture of ...	Two Gaussian distributions

From the previous results, we already expect to have more uncertainty around $t = 1$, as was the time value where a greater MSE value was obtained when training FFNNs in Figure 6. The obtained results are depicted in Figure 9 for the same samples that we have used in previous Figures. As it can be seen in both Figure 9 and the results presented in Table 7, having a measure of uncertainty provided by BNN leads to a slight reduction in accuracy. This is noticeable in the second curve in Figure 9, where it can be seen that FFNN fits better the real curve. Even so, this reduction in accuracy is not very noticeable, as seen

in Table 7, and is the cost of providing a measure of the uncertainty of the models, as noted in [30]. However, we also highlight the importance of having a measure of uncertainty for the design process, as we already know where the BNN is unsure about the outcome, and this uncertainty guides the design process, as it provides information about regions of the neck y-moment where minimal changes in the input may cause dramatic changes in the output.

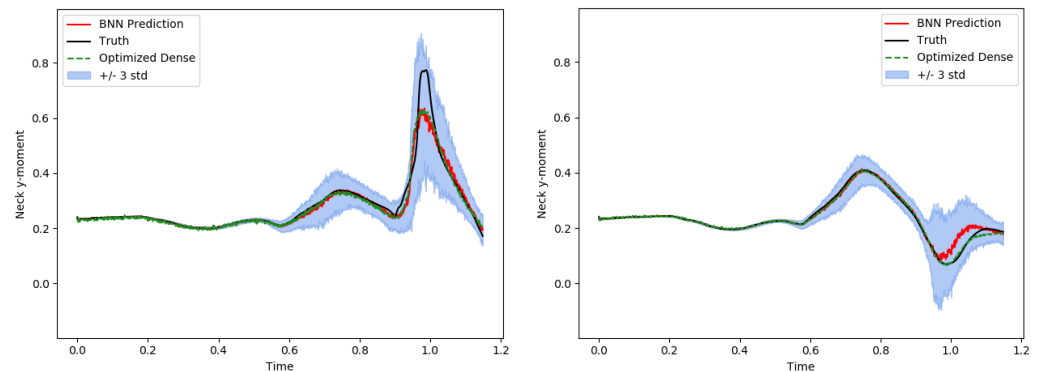


Figure 9. Comparison between predicted curves of the BNN and FFNN and the real curve chosen for two different test samples for the neck y-moment, where the average curve between the 100 predictions of the BNN is presented in red, the prediction of FFNN in green, the real values in black, and the BNN confidence interval in blue. Note that the parts where the curve becomes less predictable have a higher uncertainty in the BNN (see Figure 2), indicating the regions where the prediction is less certain.

Table 7. Evaluation metrics comparison between FFNN and BNN for the mean value of 100 predictions for the neck y-moment output curve.

Metric	FFNN	BNN
R^2	0.965	0.944
MSE	$2.331 \cdot 10^{-4}$	$3.653 \cdot 10^{-4}$
MAE	$6.669 \cdot 10^{-3}$	$8.051 \cdot 10^{-3}$

4.2.2. Resultant Head Acceleration

The parameters used to train the network for this output response are the ones already presented in Table 6, except for the number of epochs, which in this case is increased to 200, and the architecture optimized for this output response is, again, the one used before and shown in Table 4.

The obtained predictions are depicted in Figure 10, where we plot the predictions for the same two samples as in previous Figures. Using this Figure and Table 8, once again, it can be seen that there is a slight reduction in accuracy when using these networks, but a measure of uncertainty is achieved. In the first result of Figure 10, it can be observed that the BNN prediction is very far from being able to estimate the maximum value of the output curve when compared with the FFNN prediction, but the real value of the peak lies within the confidence interval predicted. Moreover, in the case that the second peak is not actually present (second result of Figure 10), the confidence interval becomes wider, while the mean of the 100 BNN predictions is quite close to the real value.

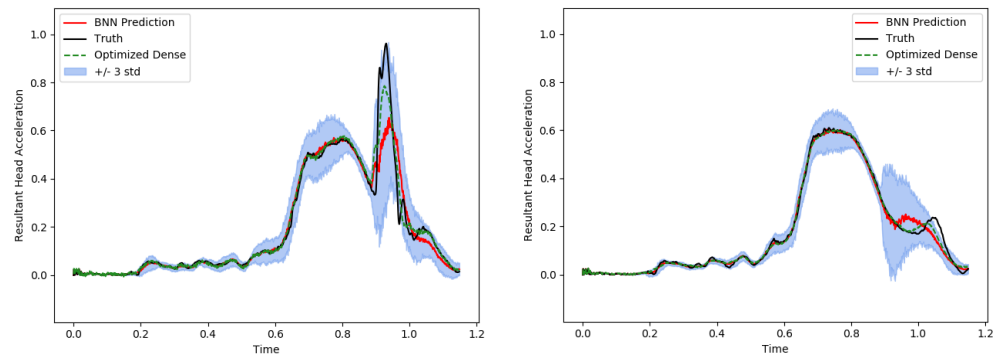


Figure 10. Comparison between predicted curves of the BNN and FFNN and the real curve chosen for two different test samples for the resultant head acceleration, where the average curve between the 100 predictions of the BNN is presented in red, the prediction of FFNN in green, the real values in black, and the BNN interval in blue. Note that the parts where the curve becomes less predictable have a higher uncertainty in the BNN (see Figure 2), indicating the regions where the prediction is less certain.

Table 8. Evaluation metrics comparison between FFNN and BNN for the mean value of 100 predictions for the Head Acceleration output curve.

Metric	FFNN	BNN
R^2	0.993	0.986
MSE	$3.153 \cdot 10^{-4}$	$6.078 \cdot 10^{-4}$
MAE	$8.543 \cdot 10^{-3}$	$1.047 \cdot 10^{-2}$

4.3. Discussion

We now proceed to summarize our results:

- We consider the problem of predicting a discrete time curve that contains information about a test dummy during a crash event. We compare the DL methods proposed with two state-of-the-art methods, SST and ERC, in order to have a benchmark. The results in Tables 3 and 5 show that FFNNs have a clear predictive advantage in all the metrics analyzed.
- It is known that FFNNs can be overconfident in their predictions [29]. In order to alleviate this problem, we also propose using BNNs, which not only give a prediction of the curve but also an uncertainty measurement. The main effect we observe is that BNNs cause a slight decrease in the performance metrics compared to FFNNs, as seen in Tables 7 and 8, but at the same time, they provide the information of which parts of the predicted curve are more certain.
- DL approaches also excel in terms of computational power and time efficiency: while FE values were obtained using 45 processing hours in a 36 CPU cluster, SST, ERC, and DL methods were run on a standard computer (Processor Intel Core i7 3820, GPU Nvidia Titan V, 16 GB of RAM), with runtime in the range of minutes: ERC training took ~ 100 s, while FFNN training took ~ 7 s.

Regarding potential limitations of our work, the dataset used is not too large, composed of 140 samples, due to the high computational load that entails generating one of these trajectories using FE simulations. Thus, it is important to ensure that the model is generalizing well; in order to do that, we follow standard practices by training using a set of samples and validating using a different set of samples, unseen during training. All the reported results are given using the validation trajectories. This is a limitation of our current work that can be addressed either by increasing the dataset size or by the use of techniques that specialize in cases of low sample size, i.e., few-shot learning techniques.

We also highlight the importance of having an uncertainty prediction by using BNNs. As commented, NNs are known to be overconfident in their prediction [29], which can

be mitigated by having not only an estimate of the trajectory, but also an estimation of its variance. Uncertainty has a key role in engineering [38], and in our problem, it means that a designer can obtain information regarding where the predictor is unsure about the trajectory, as well as information such as the maximum and minimum expected values of the trajectory (which in our cases are acceleration experimented by the test dummy).

To sum up, we provide two different approaches: FFNNs provide a slightly better prediction, at the risk of overconfidence, whereas BNNs provide uncertainty estimates as well. We consider that both approaches are complementary and can be used to provide a very accurate first approximation for the prediction problem. Note that the low computational load of DL methods can be exploited also to make a first approximation of crash responses, which can later be refined (and validated) by making use of other tools, such as FE. Moreover, resulting models can then be used for optimization, sensitivity analysis, or uncertainty analysis, among many other contexts [39,40].

5. Conclusions

In this work, we have investigated the feasibility of applying different DL models to approximate the curves of interest to predict passenger injuries in a crash event so that the design process of passive safety elements in vehicles can be significantly accelerated compared to standard FE simulations. To this end, we proposed to implement two DL models: FFNNs and BNNs, in addition to two other multi-output regression methods, which served as baseline.

The results obtained are very positive: both FFNNs and BNNs provide a very high prediction accuracy; BNNs can also provide an uncertainty estimate; and there are significant savings in computational load and time when compared to SST, ERC, and especially FE. Thus, note that the combination of high accuracy and low training time makes our DL proposal a potential game changer in the design process of passive security elements. That is why this tool is ready to use within the company's context, as it already gets good results to provide reliable first approximations on car crash responses.

This promising work also allows for several other future lines of research. The first one has to do with the number of samples: in this work, we have used only 140 samples. It is possible to gather more sequences, but it is also possible to use Deep Generative Models [41] to generate synthetic data in order to increase the number of data available. In addition to that, instead of considering only data coming from the FE simulator, data coming from real car crash tests could also be added and weighted so that the networks would give more relevance to them than to the simulated values. Furthermore, it is also proposed to optimize FFNNs using fewer training samples, investigating what is the limit number of samples with which good results continue to be obtained due to the high cost of generating these samples. Another possible research direction, related to this idea, is the use of few-shot learning techniques [42,43] in order to test whether we can further reduce the minimal number of samples needed for learning. It could also be possible to test other machine learning models different from the ones proposed in this paper, such as Support Vector Machines or Random Forests, to mention two. And finally, it is also important to test the generalization abilities of the models proposed, even out of the training distribution, as shown in [44].

Author Contributions: Conceptualization: M.L.N., J.S.J., P.A.A., J.P., S.Z. and M.G.; Methodology: M.L.N., J.S.J., P.A.A., J.P., S.Z. and M.G.; Formal analysis and investigation: M.L.N., J.S.J., P.A.A., J.P., S.Z. and M.G.; Writing—original draft preparation: M.L.N.; Writing—review and editing: M.L.N., J.S.J., P.A.A., J.P., S.Z. and M.G.; Resources: S.Z. and M.G.; Supervision: J.P., S.Z. and M.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this paper is published along with the code in the following link: https://github.com/marlahozn/DL_for_vehicle_safety (accessed on 2 September 2024). The data published has been anonymized to fulfill BMW Group disclosure policies.

Acknowledgments: We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Conflicts of Interest: Author Jonas Siegfried Jehle was employed by the company BMW Group. The remaining authors declare that the re-search was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. European Commission. *EU Road Safety Policy Framework 2021–2030—Next Steps towards “Vision Zero”*; Technical report; Mobility and Transport; European Commission: Brussels, Belgium, 2019.
2. WHO. *Global Status Report in Road Safety*; Technical report; World Health Organization (WHO): Geneva, Switzerland, 2018.
3. Road Safety Statistics 2023 in More Detail. Available online: https://transport.ec.europa.eu/background/road-safety-statistics-2023_en (accessed on 1 October 2024).
4. Ghandour, A.J.; Hammoud, H.; Al-Hajj, S. Analyzing Factors Associated with Fatal Road Crashes: A Machine Learning Approach. *Int. J. Environ. Res. Public Health* **2020**, *17*, 4111. [[CrossRef](#)]
5. Sholevar, N.; Golroo, A.; Esfahani, S.R. Machine learning techniques for pavement condition evaluation. *Autom. Constr.* **2022**, *136*, 104190. [[CrossRef](#)]
6. Komol, M.M.R.; Hasan, M.M.; Elhenawy, M.; Yasmin, S.; Masoud, M.; Rakotonirainy, A. Crash severity analysis of vulnerable road users using machine learning. *PLoS ONE* **2021**, *16*, e0255828. [[CrossRef](#)]
7. Vinta, S.R.; Rajarajeswari, P.; Kumar, M.V.; Kumar, G.S.C. BConvLSTM: A deep learning-based technique for severity prediction of a traffic crash. *Int. J. Crashworth.* **2024**, 1–11. [[CrossRef](#)]
8. Khan, M.N.; Das, A.; Ahmed, M.M. Prediction of truck-involved crash severity on a rural mountainous freeway using transfer learning with resnet-50 deep neural network. *J. Transp. Eng. Part A Syst.* **2024**, *150*, 04023131. [[CrossRef](#)]
9. Sattar, K.; Chikh Oughali, F.; Assi, K.; Ratrout, N.; Jamal, A.; Masiur Rahman, S. Transparent deep machine learning framework for predicting traffic crash severity. *Neural Comput. Appl.* **2023**, *35*, 1535–1547. [[CrossRef](#)]
10. Li, Y.; Yang, Z.; Xing, L.; Yuan, C.; Liu, F.; Wu, D.; Yang, H. Crash injury severity prediction considering data imbalance: A Wasserstein generative adversarial network with gradient penalty approach. *Accid. Anal. Prev.* **2023**, *192*, 107271. [[CrossRef](#)] [[PubMed](#)]
11. Niyogisubizo, J.; Liao, L.; Sun, Q.; Nziyumva, E.; Wang, Y.; Luo, L.; Lai, S.; Murwanashyaka, E. Predicting crash injury severity in smart cities: A novel computational approach with wide and deep learning model. *Int. J. Intell. Transp. Syst. Res.* **2023**, *21*, 240–258. [[CrossRef](#)]
12. Broughton, J. The benefits of improved car secondary safety. *Accid. Anal. Prev.* **2003**, *35*, 527–535. [[CrossRef](#)] [[PubMed](#)]
13. Abdullah, N.; Sani, M.; Salwani, M.; Husain, N. A review on crashworthiness studies of crash box structure. *Thin-Walled Struct.* **2020**, *153*, 106795. [[CrossRef](#)]
14. Santos, K.; Silva, N.M.; Dias, J.P.; Amado, C. A methodology for crash investigation of motorcycle-cars collisions combining accident reconstruction, finite elements, and experimental tests. *Eng. Fail. Anal.* **2023**, *152*, 107505. [[CrossRef](#)]
15. Idrees, U.; Ahmad, S.; Shah, I.A.; Talha, M.; Shehzad, R.; Amjad, M.; Koloor, S.S.R. Finite element analysis of car frame frontal crash using lightweight materials. *J. Eng. Res.* **2023**, *11*, 100007. [[CrossRef](#)]
16. Marzougui, D.; Brown, D.; Park, H.; Kan, C.; Opiela, K. Development & validation of a finite element model for a mid-sized passenger sedan. In Proceedings of the 13th International LS-DYNA Users Conference, Dearborn, MI, USA, 8–10 June 2014; pp. 8–10.
17. Ibrahim, H.K. Design Optimization of Vehicle Structures for Crashworthiness Improvement. Ph.D. Thesis, Concordia University, Montreal, QC, Canada, 2009. Available online: <https://spectrum.library.concordia.ca/id/eprint/976529/> (accessed on 2 September 2024).
18. Phellan, R.; Hachem, B.; Clin, J.; Mac-Thiong, J.M.; Duong, L. Real-time biomechanics using the finite element method and machine learning: Review and perspective. *Med. Phys.* **2021**, *48*, 7–18. [[CrossRef](#)] [[PubMed](#)]
19. Deck, C.; Bourdet, N.; Trog, A.; Meyer, F.; Noblet, V.; Willinger, R. Deep learning method to assess brain injury risk. *Int. J. Crashworthiness* **2023**, *28*, 760–769. [[CrossRef](#)]
20. Salazar, F.; Hariri-Ardebili, M.A. Coupling machine learning and stochastic finite element to evaluate heterogeneous concrete infrastructure. *Eng. Struct.* **2022**, *260*, 114190. [[CrossRef](#)]
21. Nastos, C.; Komninos, P.; Zarouchas, D. Non-destructive strength prediction of composite laminates utilizing deep learning and the stochastic finite element methods. *Compos. Struct.* **2023**, *311*, 116815. [[CrossRef](#)]
22. Jimenez-Martinez, M. Artificial Neural Networks for Passive Safety Assessment. *Eng. Lett.* **2022**, *30*, 1.
23. Borchani, H.; Varando, G.; Bielza, C.; Larranaga, P. A survey on multi-output regression. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2015**, *5*, 216–233. [[CrossRef](#)]

24. Spyromitros-Xioufis, E.; Tsoumakas, G.; Groves, W.; Vlahavas, I. Multi-target regression via input space expansion: Treating targets as inputs. *Mach. Learn.* **2016**, *104*, 55–98. [CrossRef]
25. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: <http://www.deeplearningbook.org> (accessed on 2 September 2024).
26. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
27. Haykin, S.S. *Neural Networks and Learning Machines*; Pearson Education: Hoboken, NJ, USA, 2009.
28. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference for Learning Representations, San Diego, CA, USA, 7–9 May 2015; Volume 7–9.
29. Huang, J.; Park, S.; Simeone, O. Calibration-aware bayesian learning. In Proceedings of the 2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP), Rome, Italy, 17–20 September 2023; IEEE: New York, NY, USA, 2023; pp. 1–6.
30. Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; Wierstra, D. Weight uncertainty in neural network. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; PMLR: Birmingham, UK, 2015; pp. 1613–1622.
31. Werbos, P. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **1990**, *78*, 1550–1560. [CrossRef]
32. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
33. Vinyals, O.; Toshev, A.; Bengio, S.; Erhan, D. Show and tell: A neural image caption generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3156–3164.
34. Hossain, M.Z.; Sohel, F.; Shiratuddin, M.F.; Laga, H. A comprehensive survey of deep learning for image captioning. *ACM Comput. Surv.* **2019**, *51*, 118. [CrossRef]
35. Zhang, M.; Yang, Y.; Zhang, H.; Ji, Y.; Shen, H.T.; Chua, T.S. More is Better: Precise and Detailed Image Captioning Using Online Positive Recall and Missing Concepts Mining. *IEEE Trans. Image Process.* **2019**, *28*, 32–44. [CrossRef]
36. Pumperla, M. Hyperas. Keras + Hyperopt: A very Simple Wrapper for Convenient Hyperparameter Optimization. 2020. Available online: <https://github.com/maxpumperla/hyperas> (accessed on 2 September 2024).
37. Saxena-Mayur; Kollnig, K.; Delaney, L.; Couairon, G. Reproduction—Weight Uncertainty in Neural Networks. 2019. Available online: <https://github.com/saxena-mayur/Weight-Uncertainty-in-Neural-Networks> (accessed on 2 September 2024).
38. Kochenderfer, M.J. *Decision Making Under Uncertainty: Theory and Application*; MIT Press: Cambridge, MA, USA, 2015.
39. Jehle, J.S.; Lange, V.A.; Gerdtts, M. Enabling the evidence theory through non-intrusive parametric model order reduction for crash simulations. In Proceedings of the 9th International Workshop on Reliable Engineering Computing (REC2021), Virtually, 17–20 May 2021.
40. Jehle, J.S.; Lange, V.A.; Gerdtts, M. Proposing an Uncertainty Management Framework to Implement the Evidence Theory for Vehicle Crash Applications. *Asce-Asme J. Risk Uncertain. Eng. Syst. Part B Mech. Eng.* **2022**, *8*, 021204. [CrossRef]
41. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
42. Wang, Y.; Yao, Q.; Kwok, J.T.; Ni, L.M. Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv.* **2020**, *53*, 63. [CrossRef]
43. Parnami, A.; Lee, M. Learning from few examples: A summary of approaches to few-shot learning. *arXiv* **2022**, arXiv:2203.04291.
44. Ye, N.; Li, K.; Hong, L.; Bai, H.; Chen, Y.; Zhou, F.; Li, Z. Ood-bench: Benchmarking and understanding out-of-distribution generalization datasets and algorithms. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 21–24 June 2022; Volume 1, pp. 7947–7958.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.