CHALMERS
UNIVERSITY OF TECHNOLOGY

Predicting Finite Element Simulation output using Machine Learning

A study to understand the potential of Graph Neural Networks in predicting vehicle occupant pre-crash simulation kinematics

Master's thesis in Biomedical Engineering

CHIARA ROSANNA FICHERA

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis in Biomedical Engineering

Predicting Finite Element Simulation output using Machine Learning

A study to understand the potential of Graph Neural Networks in predicting vehicle occupant pre-crash simulation kinematics

Chiara Rosanna Fichera

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Safety
Injury Prevention
Chalmers University of Technology
Gothenburg, Sweden 2022

Predicting Finite Element Simulation output using Machine Learning
A study to understand the potential of Graph Neural Networks in predicting vehicle occupant pre-crash simulation kinematics
Chiara Rosanna Fichera

Supervisor: Jobin John, Division of Vehicle Safety, Chalmers
Co-supervisor: Johan Iraeus, Division of Vehicle Safety, Chalmers
Examiner: Johan Iraeus

Master's Thesis 2022:70
Department of Mechanics and Maritime Sciences
Division of Vehicle Safety
Injury Prevention
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simplified Hybrid III 50th percentile dummy model.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Predicting Finite Element Simulation output using Machine Learning
A study to understand the potential of Graph Neural Networks in predicting vehicle occupant pre-crash simulation kinematics
Chiara Rosanna Fichera
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Each year around 1.35 million people die in traffic crashes and many more get injured. Human Body Models have become a popular tool to understand and prevent injuries and fatalities as they simulates the human body response during crash scenarios.

With improvements in the accuracy and bio-fidelity of Human Body Models, the demand for computational resources is increased up to requiring days for a presingle crash simulation. Therefore, Dimensionality Reduction methods can be used to create a surrogate that is a smaller representation of the model. The reduced representation translated to a latent space that still retains the complexity and the features of the simulation. To build the surrogate, supervised learning tasks have been implemented on the reduced dimension to map the simulation to its parameters.

To create a compressed version of the input simulation, Graph Neural Networks have been considered since they incorporate the geometrical structure of the model. The Graph Neural Network has been compared to an equivalent Convolutional Neural Network architecture and to a Principal Component Analysis. Random Forest, Gradient Boosting, and XGBoost were chosen and compared to build the surrogate model. Due to computational limitations, and since pre-crash simulations are already time-consuming, a simplified Hybrid III dummy model was chosen in place of a full Human Body Model.

For all Dimensionality Reduction methods, the accuracy of the reconstruction improves by increasing both the number of samples and the latent space size. Principal Component Analysis shows a better performance in terms of lower errors compared to Graph Neural Network. Moreover, the Graph Neural Network structure is comparable to an equivalent Convolutional Neural Network architecture in term of performance. Random Forest and Gradient boosting proved to be better than XGBoost. Principal Component Analysis requires less computational time and resources.
Although Graph Neural Network was outperformed in this study, further improvements in the development of the method may still have potential.

Keywords: HBM, Dimensionality Reduction, Graph Neural network, Principal Component Analysis, Machine Learning.

Acknowledgements

I would like to thank my supervisors, Johan and Jobin, for all the inputs during the weekly meetings.
A thanks to Alex and Julia for the stimulating discussion on the topic during the spring.
A Luca, per avermi incoraggiato fin dall'inizio senza alcuna riserva per poter vivere e fare esperienza di questo sogno nel cassetto.
Un grazie speciale alla mia famiglia, e soprattutto alla mamma, per fare il tifo per me e per avermi permesso di vivere quest´avventura unica. E alla nonna, che spera sempre io possa tornare a casa.

A loving thanks to Thomas, the best supporter ever, for encouraging me all the way and for believing in me, without a doubt, from the very first moment. Words cannot express enough gratitude for you.

Chiara Rosanna Fichera, Gothenburg, August 2022

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence.
ANN Artificial Neural Network.
ASIRT Association for Safe International Road Travel.

CNN Convolutional Neural Network.

DR Dimensionality Reduction.

FE Finite Elements.
FE-HBMs Finite Elements - Human Body Model.
FEM Finite Elements Methods.
FFNN Feed Forward Neural Network.

GNN Graph Neural Network.

HBM Human Body Model.

MAE Mean Absolute Error.
ML Machine Learning.
MSE Mean Square Error.

NN Neural Network.

PCA Principal Component Analysis.

RMSE Root Mean Square Error.

1
Introduction

Finite Elements simulations are used in automotive development to predict and investigate injuries, as they represent an useful tool to save resources in crash test simulations. However, producing many simulations is computationally demanding and thus time-consuming.
Machine learning is an innovative method applied in many fields to predict states based on a large quantity of data. Here, this method is applied to a Finite Element (FE) model representing a human body in a pre-crash scenario to reduce computing time and increase efficiency. These models aid to study crash kinematics and understand injuries from vehicle crashes to find ways to mitigate and prevent these injuries.

1.1 Background

According to the *Annual Global Road Crash Statistics* of the Association for Safe International Road Travel (ASIRT) around 1.35 million people die each year due to traffic accidents [2]. Furthermore, every year between 20 and up to 50 million suffer from non-fatal injuries. Even though, Europe has the lowest road-traffic mortality rate with a tendency to fall, it is still a leading external cause of death and injury.
Sweden's fatality rate in traffic is one of the lowest in the EU (27 per million population in 2015) [3]. The total cost of road accident casualties (both fatalities and injuries) is estimated to be 48,5 billion euros in the European Union [3]. The aim of traffic safety research is to decrease the number of accidents in general but especially fatalities and injuries. As a very idealistic ambition, the Vision Zero aims to eliminate all traffic fatalities and severe injuries by increasing the safety [2, 5]. The goal is to have a holistic approach towards traffic safety including traffic environments, injuries and fatalities prevention, dynamics of car crashes as well as designing new safety systems.
In the automotive field, active and passive safety are the two different branches that focus on car safety and crashes. In active safety, new features (e.g. driver assistant) are designed to prevent the crash to occur. Parallelly, passive safety investigates and develops measures to mitigate the seriousness of the crash.
In this regard, the focus of injury prevention is to assess and reduce the injury severity of the vehicle's occupants and to understand the mechanical response during the impact. Commonly, these findings can be used to develop new safety features (e.g., roof-rail airbags) or improve existing ones (e.g., seat belt pre-tensioners). Moreover, analyzing the few moments before a crash can give a better understanding of what

can be done and improved before the crash happens. During the pre-crash phase, the driver might undertake manoeuvres or adjustment to avoid the impact. These lead the driver to be out of his ideal position and resulting in more severe injuries during the crash. This behaviour occurs in an higher extend in case of automated driving where the driver is partially or completely disengaged from the driving activity and therefore unaware of the forthcoming maneuvers.

1.1.1 The crash phase

In order to understand the dynamic of an impact, a brief explanation of the different phases of a crash follows. The phases are shown in the Figure 1.1. A car crash event can be split into three different phases: pre-crash, in-crash and post-crash [6].
In the pre-crash phase, the chance of a crash to occur is high. In this time frame, usually around the order of seconds, the accident can be avoided, or the severity reduced, if actions are taken. At the end of this phase, if the crash cannot be avoided, a point of no return is reached - as shown in Figure 1.1. Usually, the main cause of car crashes is human error but external factors such as road conditions can also contribute substantially [7].
The next step is the in-crash phase which lasts around 150 ms. In this stage, the impact occurs. Nothing in this phase can be done to avoid the impact. The occupant of the vehicle is subject to forces that can cause severe injuries if the delta velocity (the change of velocity during the impact) is high and the occupant comes in contact with the inner parts of the vehicle.
Lastly, the post-crash phase follows the in-crash phase. Both the vehicle and the occupants are halted and rescue takes place. In this stage, the severity of the crash can be assessed and fatality risk can be calculated.

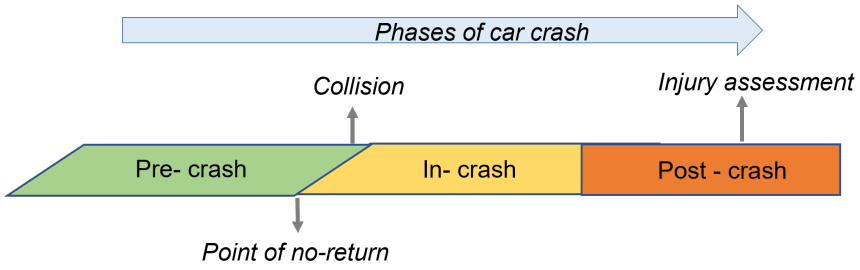


Figure 1.1: Events occuring during crash phase.

1.1.2 The evolution of test subjects

To better understand the dynamics of the impact, crash scenarios are reproduced using a human substitute as vehicle occupant. At early stage, volunteers, animals, and human cadavers were used as subjects to conduct studies [4]. In the 1950s, the ethical concerns along with anatomical differences of the subject moved the focus to develop mechanical dummies. A dummy, as shown in Figure 1.2, is a mechanical substitute that resembles the human body during tests and is able to record the forces and deformations during an impact [4].

Figure 1.2: Hybrid III 50th percentile male dummy model used for frontal impact test.

At first, crash test dummies were incorporated in FE simulation environments to run crash tests in a virtual environment. This was done to reduce the effort and cost of destructive physical tests and allows more extensive testing of different crash scenarios with a wider variation of crash parameters.
Thanks to the high precision of the FE simulations, it is possible to investigate the local deformation and the strains. At a later stage, Finite Element - Human Body Models (FE-HBMs) have been developed (see example in Figure 1.3) to reach higher bio-fidelity of the human body and yield a more accurate prediction of injuries. In most recent research, Active-HBMs have been developed and used to incorporate the response of muscles activation in the model [39, 40]. In addition, morphing has been used to shape and adapt varying or outlying anthropometric features on HBMs [41].

1.2 Motivation

Pre-crash simulations are computationally expensive and time-consuming. Simulation time through FE solver of pre-crash scenarios is extremely long due to the long duration of the event. State-of-the-art FE models are based on complex Partial Differential Equations, which involve a high number of complex variables and boundary conditions.Hence, obtaining a reduced representation of complex data, also called surrogate or meta-models, can be a solution for such challenges.
Dimensionality Reduction entails having a surrogate model that retains most of the complexity and features of the model. DR reduces the number of features to a small number of meaningful components. It is convenient when working with big and complex models that have many redundant features. As result, it is possible to obtain a clear and straightforward data visualization. Having a smaller model

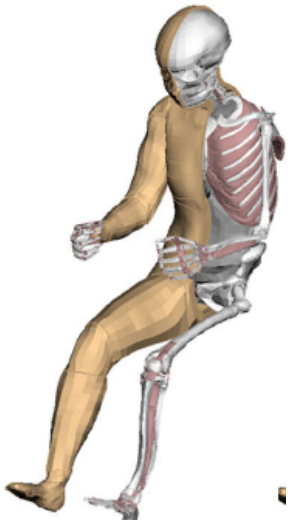


Figure 1.3: HBM v.9 Female model developed by SAFER

requires less computational resources, having reduced training and computational time.

The dimensionality reduction process is presented in Figure 1.4.

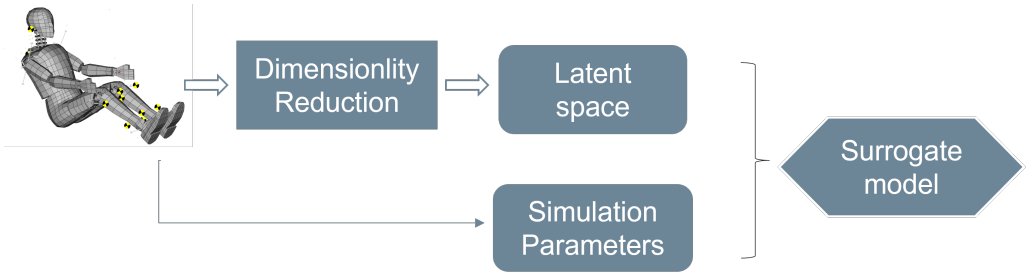


Figure 1.4: Surrogate model process

Specifically, it can be divided into two main parts:

- building a low dimensional input, also called latent space, where the high dimensional input is encoded
- create a surrogate modeling on the reduced dimension model by mapping the model parameters to the simulation. The model parameters (e.g., breaking or steering acceleration) rule and influence the simulations outcomes.

These two steps are obtained by using ML algorithms. The first step is an unsupervised learning task, whereas the second is a supervised task. In this project, different unsupervised methods are used to encode the input in a lower space.

1.3 Aim and limitations

This project aims to implement a Dimensionality Reduction method based on Graph Neural Network, to build and evaluate a surrogate model that can predict kinematics of the pre-crash phase using input data of a Human Body Model model. As a consequence, a model that predicts the kinematics history of a pre-crash simulation can be used as a substitute for FE-solver-based simulations.

Below follows a list specifying the research objectives for this thesis.

- Identify and investigate a GNN architecture for Dimensionality Reduction
- Compare the GNN architecture to an equivalent CNN
- Compare the GNN to Principal Component Analysis
- Evaluate different supervised methods to build the surrogate model
- Analyze the time consumption and the computational resources of the methods analyzed compared to an FE solver

Due to limited computational resources a Hybrid III 50th percentile Dummy model was used instead of the SAFER HBM v9.

2
Theory

In this chapter, a brief introduction on the methods used in the project such as Finite Elements Methods (FEM), Dimensionality Reduction methods, Graph Neural Network (GNN), Principal Component Analysis (PCA) and supervised learning algorithms is given.

2.1 Finite Elements Methods

Finite Elements Methods are a numerical method that overcomes the difficulty of resolving complex Differential Equations (DEs) with analytical approaches. The analytical approach requires the assumption of many hypotheses if the set of equations can be solved for the global system. Thus, the unfeasibility to investigate accurately the local effects. In the numerical approach, the space is considered as a collection of subdomains - that form a mesh - called Finite Elements - illustrated in Figure 2.1. For every Finite Element, a set of coupled equations is formulated and an approximate numerical solution can be found.

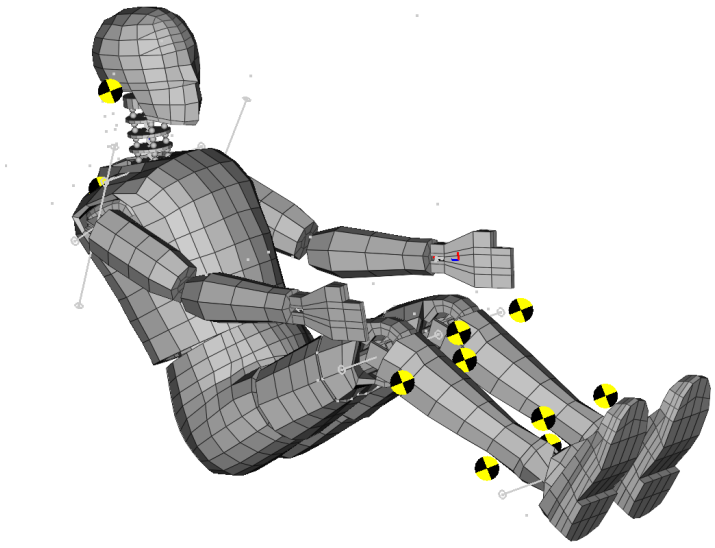


Figure 2.1: Hybrid III 50th percentile FE Dummy model

One of the simplest approaches to develop a numerical solution is based on *Euler Methods*. A simple and generic case is presented to show how it is formulated. For a given equation in the time domain, the initial condition for $t = t_0$ is known, as shown in 2.1.

$$\begin{cases} \dot{x}(t) = f(x,t) \\ x(t_0) = x_0 \end{cases} \tag{2.1}$$

Let´s consider the nth time-step for $x(t = t_n)$. The equation will be written as follow:

$$\dot{x}_n = f(x_n, t_n) \tag{2.2}$$

The derivative of $\dot{x}_n$ is defined also known as Euler approximation:

$$\dot{x}_n \approx \frac{x_{n+1} - x_n}{t_{n+1} - t_n} = \frac{x_{n+1} - x_n}{\Delta t} \tag{2.3}$$

From Equation 2.2 and 2.3, it is possible to obtain:

$$\frac{x_{n+1} - x_n}{\Delta t} = f(x_n, t_n) \tag{2.4}$$

Rewriting the equation and isolating the term x_{n+1}:

$$x_{n+1} = x_n + \Delta t f(x_n, t_n) \tag{2.5}$$

The Equation 2.5 takes the name of *Euler explicit method*. Since it is obtained from known terms, x_n and $f(x_n, t_n)$ and the next iteration is isolated on one side of the equation. This formula approximates the next value of the function by knowing the previous step values and the step size taken in the tangential direction. The geometrical meaning of the approximation approach is shown in Figure 2.2 [42]. The Euler approximation is derived from the truncated Taylor expansion of the derivative definition and therefore has an error of $O(h^2)$.

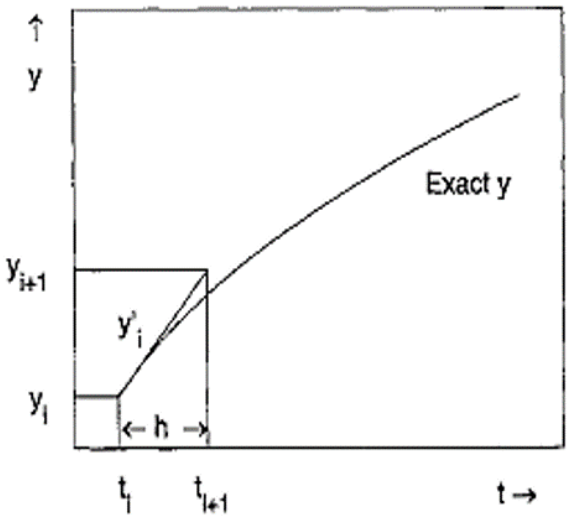


Figure 2.2: Single step Euler approximation

Alternatively another formulation can be drawn. By re-writing the 2.3 and considering the previous time-steps we would have:

$$\dot{x}_n = \frac{x_n - x_{n-1}}{\Delta t} \tag{2.6}$$

And obtaining:

$$x_n = x_{n-1} + \Delta t f(x_n, t_n) \tag{2.7}$$

And then, re-indexing from x_n to x_{n+1}:

$$x_{n+1} = x_n + \Delta t f(x_{n+1}, t_{n+1}) \tag{2.8}$$

The Equation 2.8 shows the *implicit or backward Euler formula*. It is important to notice that in this version, the term $n + 1$ appears on both sides of the equation, therefore knowing the current step is not sufficient to solve the equation.

The implicit and explicit Euler formulas are two different methods to approximate and solve a differential equation. Numerical solvers can be either based on the explicit or the implicit method. The implicit methods are generally used for static and dynamic analyses when investigating linear or quasi-linear effects [43]. However, multiple iterations and matrix inversion are needed to solve the equations. The approach is computationally heavy on memory. In contrast, the explicit method, due to its iterative formulation, is lighter on memory usage and suited for highly non-linear geometrical and material effects. However, since the methods can be unstable, to ensure convergence small steps need to be considered, making it suitable only for short simulations [43].

Overall, there are different trade-offs between choosing an implicit or explicit solver. The method of the solver is chosen based on the field of application. In this project, an explicit solver like LS-DYNA is more suitable for the application, due to the high non-linearity and short duration of crash simulations.

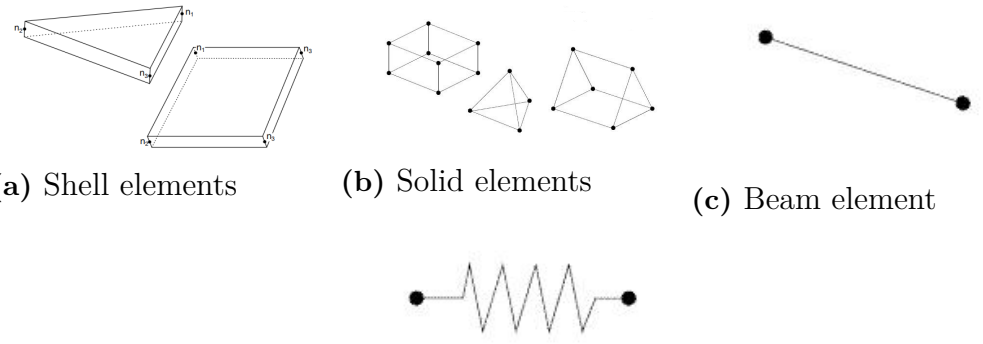
The mesh

The fundamental blocks that constitute a mesh are shell, solid, discrete, and beam shells. The different shell parts are shown in Figure 2.3 below.

In particular:

- Shell elements are elements composed of 4 nodes, or vertexes.
- Solid elements are defined by 8 nodes. They can be tetrahedrons, pentahedrons, or hexahedrons.
- Beam elements are defined by 3 nodes, where the first 2 describe the geometry and the third the orientation.
- Discrete elements are defined by 2 nodes. It can be either a spring or a damper between two nodes.

In addition, there is one more category called constrained parts or elements. The constrained category applies constraints within the model and its structural part. The distance between two constrained nodes is fixed and does not vary throughout the simulation. They can be used to model a joint, for instance, or to impede the degree of freedom of parts.



(a) Shell elements **(b)** Solid elements **(c)** Beam element

(d) Discrete elements

Figure 2.3: Different FE elements [37]

2.2 Machine Learning

In recent years, Machine Learning (ML) has gained importance and found application in many fields. Machine Learning is a sub-field of Artificial Intelligence (AI) that aims to use data and algorithms to imitate the way that humans learn. This entails the human ability to improve itself over time [9].
ML can be classified into different categories based on the learning rule - supervised or unsupervised. A supervised algorithm uses labeled data during the training process to validate the prediction made. It requires a 'supervisor' to determine and label the data before being used in the training process. Supervised models are generally used for classification or regression tasks. In contrary, unsupervised models do not require labels but seek patterns in the dataset itself. They are typically used for dimensionality reduction, clustering, and association tasks [18]. Often, these two algorithms are based on Artificial Neural Network (ANN)s.

2.2.1 What is a Neural Network?

The name Neural Network comes from the neuron which is a biological element of a brain network. In the brain, a neuron receives and processes inputs and eventually transmits output signals to other neurons. ANNs are designed inspired by this physiological process.
Similarly, the building block of an ANN is called a neuron, which forms an intricate interconnected structure that can be extended up to millions of nodes, as we could find in a brain. The similarity between a biological and artificial neuron are highlighted in Figure 2.4 [1].
As shown in the Figure 2.4, a weight is associated with every input. In the case of multiple inputs, the neuron outputs a weighted sum of the inputs.
To continue the process, the output of the first layers of neurons is propagated to the next layer and so on [16]. This structure, as shown in 2.5, takes the name of Feed Forward Neural Network (FFNN), as the output is always forward-propagated to the next layer. The layers located between the input and output are defined as

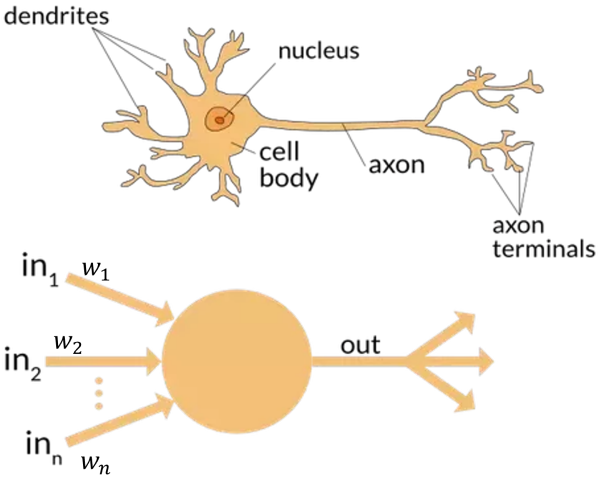


Figure 2.4: Structure of biological and artificial neuron [1].

hidden layers. FFNN, shown in Figure 2.5 takes also the name of fully connected layers, meaning that each node of a layer is connected to all the units of the previous and next layer.

The learning process is driven by different parameters. An ANN can have trainable parameters (e.g, the connection between the layers) that are adjusted during the training process. In contrast, the other category of parameters, called hyperparameters, control the training process and are set beforehand (e.g., learning rate, epochs, number of layers, and many others).

ANNs learn tasks from empirical samples or data during the training phase. No previous knowledge or theory is used by the model to learn, and therefore requires a considerable amount of data.

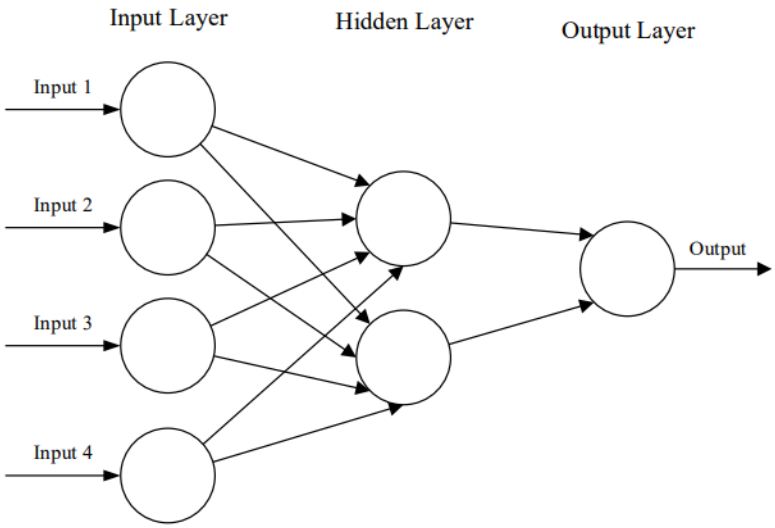


Figure 2.5: An example of a Feed-forward Neural Network composed of three layers [17].

The training process updates the weights at every cycle, or epoch, based on the performance of the network. The weights are adjusted to minimize the cost function or loss. The loss is a function that relates the values of the model at a k-step to the k-target value. A low loss, generally indicates that the model prediction is close to the ground truth.

2.2.2 Convolutional Neural Networks

The structure of fully connected layers is not applicable to big inputs, such as images, as they need to store connections for every pixel. In addition, the network does not have any spatial information. To overcome this challenge, CNNs have been developed and become widely applied in the computer vision field. Convolutional Neural Networks are a type of ANN that uses the convolution operation. The idea is to consider the data structure as a grid and to share information within a smaller area, called receptive field. An image is a great example of a grid structure, where every pixel of the image represents one node of the grid structure. The convolution operation extracts meaningful features from the data structure. The convolution starts with a kernel, which is a small matrix of learnable parameters, or weights. In Figure 2.6, the convolution process in its different phases is shown. The kernel slides over a 2D plane and computes element-wise product between the weight and the pixel (see Subfigure a and b of 2.6). The weighted product is then summed into an output pixel. The process is repeated for the entire grid structure. By sliding over the input plan, the convolution process results in a feature map, which is a weighted sum of the input features and a smaller representation of the input (as shown in Figure 2.6c).

2.2.3 Graph Convolutional Networks

To first introduce Graph Convolutional Networks a few steps need to be taken. First, a definition of a graph will be presented, followed by what a GNN is and lastly how convolution is performed.

What is a graph?

In a formal definition, a graph, shown in Figure 2.7a, is an object composed of *nodes* (see Subfigure 2.7b) that are connected by *edges* (see Subfigure 2.7c). The connections indicate a relationship between the nodes. Mathematically, a graph is expressed as:

$$G = (V, E) \tag{2.9}$$

Here, V indicates the vertices, or nodes, and E is the edge between them.
A direction of the edges can be defined when sharing the information between the nodes. Given the graph structure, the information can be either stored in the nodes or/and edges. This means that the relationships among the edges are part of the information too. To store a graph structure and its information, an adjacency matrix can be derived. The adjacency matrix is a binary matrix of size $n \times n$, where n is the number of nodes. If for example, the node j is connected to the k-node, at the

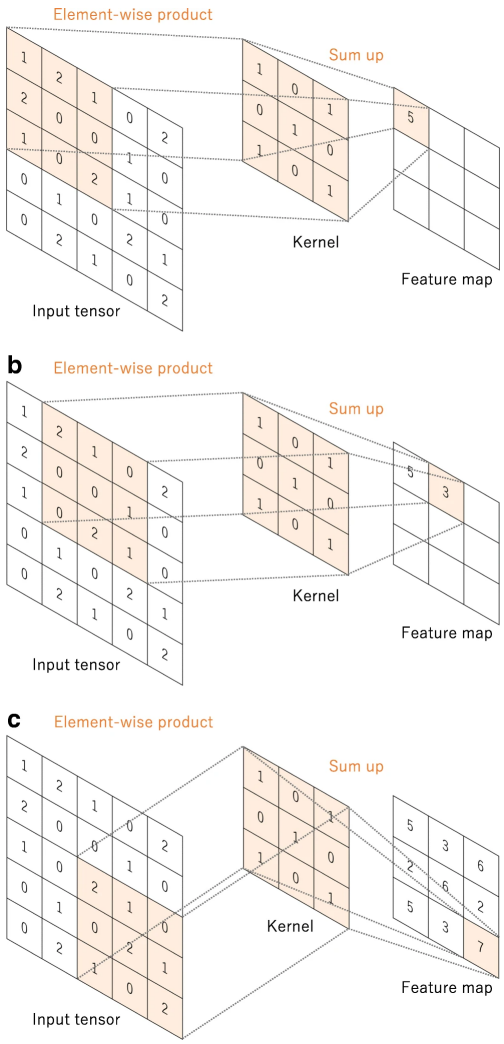
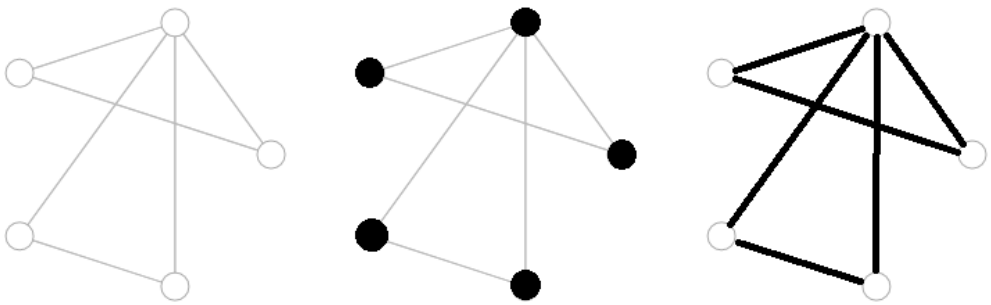


Figure 2.6: CNN convolution operation process [24].



(a) a graph with **(b)** nodes highlighted **(c)** edges highlighted

Figure 2.7: General representation of a graph with nodes and edges [20]

entrance of matrix *j-k*, *k-j* a 1 will be placed. Otherwise, if there is no connection, there will be a zero. Given this reference, the adjacency matrix is a hollow matrix (i.e. zeros on the diagonal) since a node does not have a connection to itself.

Besides the matrix, there is also another way to show the relationships among the edges. This is represented by the adjacency list. The list is of size $[2, num\_edges]$, and it stores the link between each node. For example, if node -j and -k are connected there will be a pair *[j,k]* [21].

For instance, in the following Figure 2.8, an example of a graph structure and its derived edge list are presented. The number inside the nodes indicates the label of the node. The node feature contains the information stored in the node, e.g. -1 for the node x_1 [21].

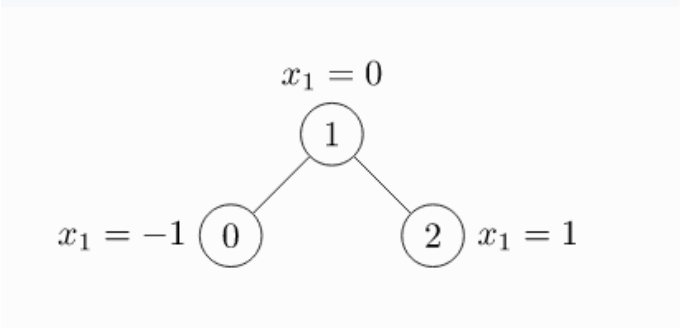


Figure 2.8: An example of a graph [21]

The adjacent list and matrix of Figure 2.8 highlighting the nodes connection would be [1]:

$$adj\_list = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 2 \\ 2 & 1 \end{bmatrix} \qquad adj\_matrix = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

A Graph neural network

A Graph Neural Network is a neural network that employs graphs-structured data [20]. The peculiarity of the graph consists of a lack of ordered grid structure, namely an un-ordered structure for the node in the graph. They work in a similar way that a neural network is working on images, for example. This application has enabled to see different experiments from another perspective. To cite a few, GNNs have a large application and success in certain areas like chemistry or social networks among others. In the former case, the edges and the nodes of a graph encode a molecule, made of atoms and bonds [23]. As for the latter, the nodes are represented by people´s profile and the edges could be the connections such as ads or people recommendations.

[1] From: https://pytorch-geometric.readthedocs.io/en/latest/ in *"Introduction by example"*

Graph Convolution

In Graph Neural Networks, convolution works differently than a standard CNN. A graph, per definition, consists of a lack of ordered structure. Compared to an image, in which the position of the pixel is unique and determined, in a graph, there is no such thing. When it comes to convolution, though, there are other ways to aggregate information. The method is called Message Passing and it was first introduced by T. Kipf and M. Welling, illustrated in Figure 2.9 [22]. From the left, given a selected node, the surrounding nodes - represented by the hollow circles - in h-hops distance are gathered and the features are aggregated. The adjacency matrix is used to gather information of the neighboring connection. The mathematical operation in which the nodes' features are aggregated is defined using the *aggregation function* and usually the sum, max, mean, etc... are used. As a result, in the next layer, on the right of Figure 2.9, the node's features are updates.

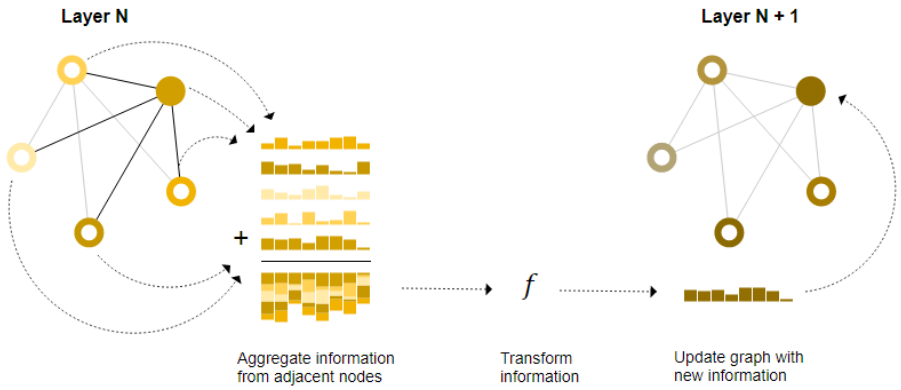


Figure 2.9: Graph convolution operation [20]

In this project, the convolution function is a GCN class [22]. The aggregation function adopted is the sum, meaning the nodes' features are summed with each other, as shown in Figure 2.9. The convolution also entails self-loops. This implies that when considering the surrounding nodes, the features of the selected node itself are also aggregated.

2.2.4 Unsupervised methods

An unsupervised neural network works with unlabeled data. Due to this characteristic, an unsupervised model is used to find similarities or differences in data, cluster the dataset, or dimensionally reduced it.
In this project, three different Dimensionality Reduction methods have been investigated: Graph Convolutional Autoencoder, Convolutional Neural Network and Principal Component Analysis (PCA).

Graph Convolutional Autoencoder

An autoencoder is a type of neural network model that encodes the input data to a latent space, a reduced representation of the original input, and then decoded it

back to the original status. One of the strengths of the autoencoder is the capacity of reconstructing the input dataset from the latent space to the original neatly. One variant of the autoencoder is the one used in this project, called convolutional autoencoder, due to the employment of convolutional layers in both the encoding and decoding part.

The architecture The autoencoder, shown in Figure 2.10 is composed of two parts: the encoder - on the left - and the decoder. The former is the part that encoded the input to the latent space, whereas the latter is used to reconstruct the input from the latent space. The latent space, also called bottleneck, is usually a smaller representation of the input.

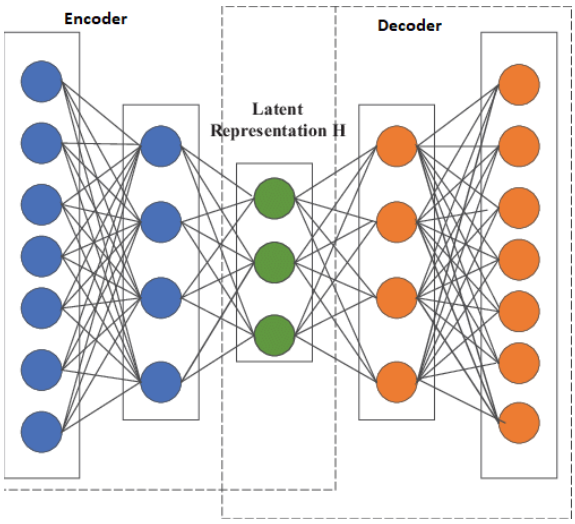


Figure 2.10: General architecture of an autoencoder [25].

Specifically, the encoder is composed of a few different building blocks: convolutional layers, a flatten layer, a pooling layer, and linear, or fully connected layers. In contrast, a decoder is composed of convolutional layers, an unflatten layer, an unpooling layer to make it consisted with the coding part, and linear layers.
The graph convolutional layer implements the Message Passing function. The flatten layer flattens a multidimensional input tensor to a 1D tensor. This layer is fundamental since the latent space is made of a 1D tensor. The unflatten layer is implemented using a reshape function to reach the opposite outcome. A pooling layer reduces the dimension of the input layer by a selected factor. The reduction of dimension is driven by the following criteria. The most common pooling layers are MaxPooling and Average Pooling. In the MaxPooling, the max feature of the kernel considered is kept and the others are discarded, whereas on the Average Pooling the average of the features considered is computed. The pooling layer forms the decoder part, that is where the features are reduced. The linear layer is a fully connected layer. In the encoder, it is implemented after the pooling layer to reduce the number of features of a layer. This layer allows the model to go from a few thousand features to only a few, forming the latent space. In the decoder, instead, it increases the number of features in the layer from the latent layer to the next one.

Principal Components Analysis

PCA is a ML method used for dimensionality reduction that geometrically projects the observation on a reduced dimensions variables called *Principal Components* (PCs). It is often used with big datasets that are computationally hard to handle. The advantage consists of having a smaller dataset that is easy to process and still contain most of the information [19].
The idea behind the method is to try to preserve as much variability of the dataset as possible [19]. This is achieved by finding new variables called PCs that are linear functions of the observations. PCA minimizes the distance between the data and the projection on the PCs. As a result, it maximizes the variance. Due to this fact, fewer elements are needed to represent the majority of the variance in the dataset. The PCs found are not correlated with each other, and they are sortes so that the first one contains the highest variance. In the figure 2.11 below, an example is given, the observation and PCs have been shown on a 2D plane. The dataset has been reduced and represented by 2 principal components (the two orthogonal vectors). The first component is the one that lies in the direction where most of the information is contained, i.e where the variance is maximed.

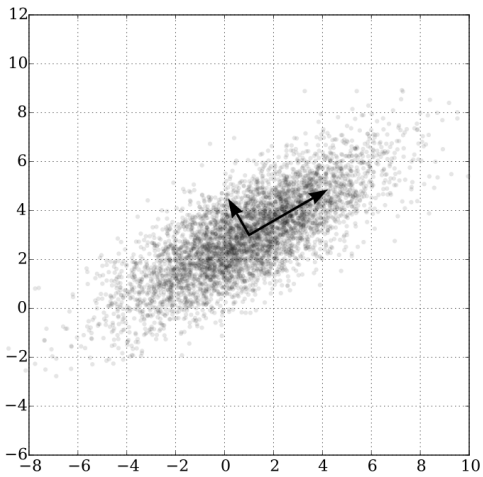


Figure 2.11: Principal component analysis on a sample dataset

To further explain the method, the process can be divided into different steps:

1. Standardize the dataset
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal components axes

Standardize the dataset The standardization, or Z-score, consists of centering the data by using the mean and the standard deviation. The transformation makes

the PCA robust to outliers. This is done by scaling the observations according to the following formula.

$$z = \frac{value - mean}{standard\_deviation} \tag{2.10}$$

Compute the covariance matrix and identify the principal components The second step consists of computing the covariance matrix of the standardized data to find the possible correlation. The covariance matrix is a symmetrical matrix of size $p \times p$, where p is the number of variables. By finding the covariance matrix, and using the eigendecomposition, the principal components are represented by the eigenvectors.
Let´s assume that inside a dataset there are p variables and n observation for each variable so that the dataset is a $p \times n$ matrix that we will call $\mathbf{X}$. X $\in p \times n$.
First, we look for the correlation within the matrix that maximize the variance. The variance will be given by:

$$a\mathbf{X} = [a_1x_1, a_2x_2, a_3x_3..a_nx_n] \tag{2.11}$$

where a represents the vector containing coefficients. The variance of the linear combination can be expressed as

$$var(aX) = a^T Sa \tag{2.12}$$

where a_n are the coefficients of the linear combination, a^T is the transpose of a and S is the covariance matrix. To solve the equation we first need to assume that we have a unit norm vector, namely $a^T a = 1$. Therefore the equation becomes:

$$aSa - \lambda(a^T a - 1) = 0 \tag{2.13}$$

By differentiating with respect to a:

$$Sa - \lambda a ==> Sa = \lambda a \tag{2.14}$$

From this equation we see that a correspond to the eigenvectors and λ to the eigenvalues of the matrix covariance S.
To give an example, the covariance matrix S for 3 variable datasets will be:

$$S = \begin{pmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{pmatrix} \tag{2.15}$$

Once the eigenvalues have been found, the higher eigenvalue corresponds to the first principal components, since it represents the most variance [19].

Create a feature vector to decide which principal components to keep In this step the dimensional reduction takes place. Once found the components and the most significant ones identified, a feature vector is created. The feature vector contains the kept eigenvectors in its columns. Usually, only some of the

components are kept, since they can explain enough variance. To give an idea, the review from Jolliffe and Cadima [19] reports that 70% of the variance is a good cut-off or, depending on the case, 2-3 PCs are enough, since most of the variance is usually contained in the first components.

Recast the data along axes of the principal components In the last step, the dataset is reoriented from the original axis to the axis identified by the principal components. This is done by multiplying the feature vector transposed with the transposed standardized dataset.

$$feature^T * dataset^T_{standardize} \tag{2.16}$$

2.2.5 Supervised methods

The supervised methods implemented in this project are regression algorithms. The regression relates a dependent variable to one or more independent variables. It is considered a supervised algorithm because the label or target are represented by the independent variables, or predictors. A simple example can be Linear regression - shown in Figure 2.12. It finds the best linear fit between two variables. The best fit possible is achieved when the residual sum, i.e. the squared distance between the point and the straight line - is minimum. The points represent the observations whereas the line is the best fit found by the algorithm [26].

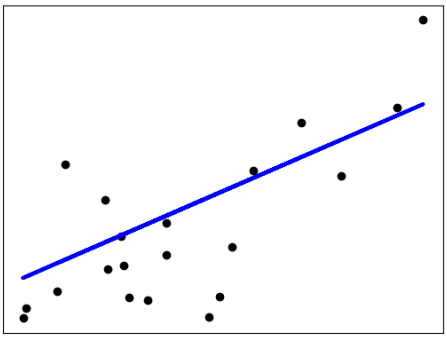


Figure 2.12: Example of Linear regression.

However, there are also other methods based om different algorithms. Ensemble methods combine different models to increase the accuracy of the results. There are three different groups of ensemble methods: bagging, boosting, and stacking.
Bagging is short for bootstrapping and aggregation. Bootstrapping is a sampling technique that uses sampling with replacement. [2] The aggregation, instead, consists of combining the individual predictions when making the final prediction. The bagging method minimizes the variance and reduces the risk of over-fitting, i.e.

[2]Sampling with replacement means that when extracting one item from the distribution, this item is placed back in the distribution before sampling again.

when the model learns the training data too well that results in learning the noise and worsen the prediction of unseen data. The Random Forest method belongs to this category. The boosting methods instead, base their predictions on learning from previous mistakes. This improves the accuracy of the prediction over time. Both gradient boosting and XGboost are boosting methods. Lastly, stacking, or stacked generalization, trains the algorithm using several bagging or boosting methods.

Random Forest

To explain the Random Forest methodology a brief introduction to Decision Trees is presented.

A decision tree is a tool that uses binary logical operation to make decisions based on the data given. Visually it is represented by a flowchart of decisions with binary outcomes, as shown in Figure 2.13. The structure is simple. The root and internal nodes are decisions. Every decision has an outcome, that can either be another decision, the internal node or a leaf node. The leaf node represents the end of the branch, namely the outcome of the branch. The depth of the tree is determined by the minimum number of questions to reach an accurate prediction.

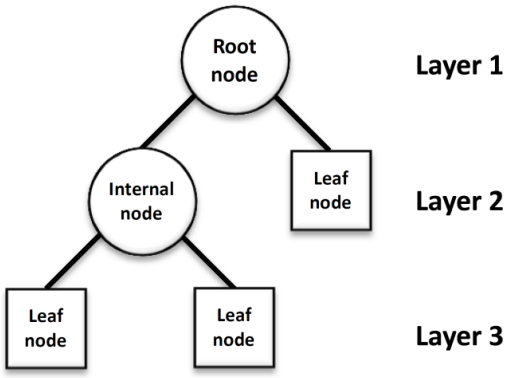


Figure 2.13: Model of a decision tree [28].

Random forest combines several classifying decision trees on various sub-samples of the dataset. During the training phase, the algorithm trains multiple decision trees parallelly. When it comes to the prediction, the result of all the trees is averaged to improve its accuracy and limit overfitting, as shown in 2.14.

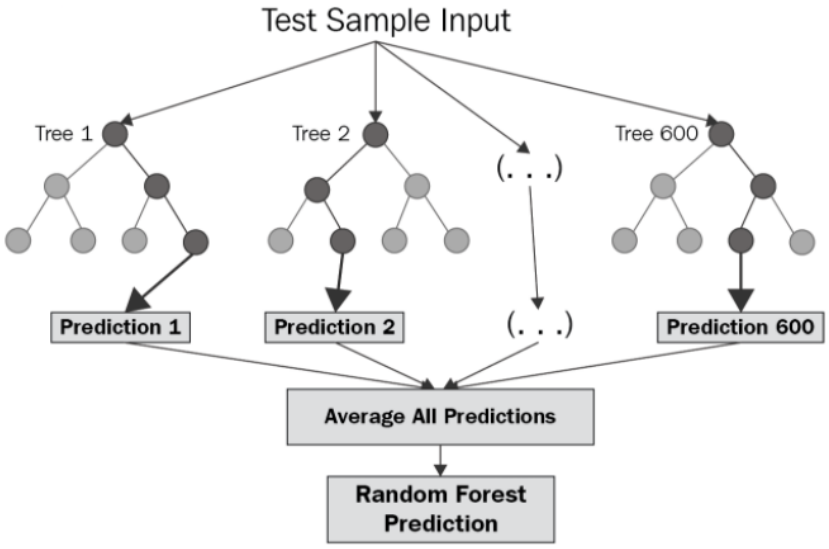


Figure 2.14: Random forest method [29].

Gradient Boosting

Gradient Boosting belongs to the boosting category. As Random Forest, it is based on decision trees. However, instead of building parallel trees, the trees are built consecutively. As the name suggests, a gradient-descent algorithm is used, which helps the algorithm to find the best fit when the gradient is minimized. In Figure 2.15, the algorithm is shown. First, it builds a tree and computes the error, i.e. the residuals. The second tree will be built based on the error of the previous one. The process continues and iterates the same steps. In this way, the error decrease over time, since the algorithm learns from previous mistakes. Moreover, there is also one important hyperparameter, the learning rate. Every tree is scaled by the learning rate. In this way, not every tree contributes in the same way, and only the more significant ones contribute to the final prediction.

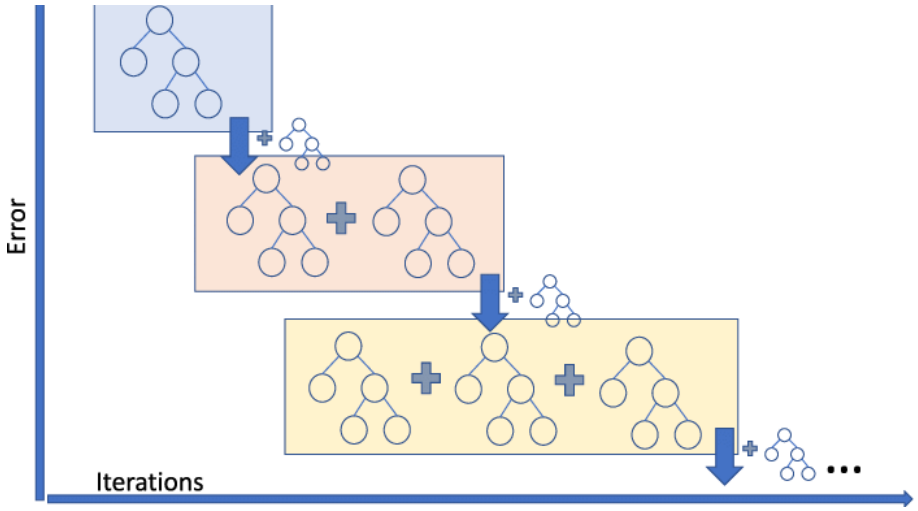


Figure 2.15: The addictive process of the Gradient boosting algorithm [30].

XGBoost

XGBoost stands for Extreme Gradient Boosting. It can be considered a development of Gradient Boosting. It is optimized to have faster computational speed and better performance. Here, the trees are built in parallel instead of consecutively.
The goal, as for the other algorithms, is to reduce the residuals, between the prediction of the tree and the target value. In the process, a few steps can be identified [32].
First, a tree is created. Then, the prediction for the first tree is made. The average of the predictions is computed and the residuals are calculated with the loss function of choice. Nextly, a similarity score is calculated as follows:

$$similarity - score = \sqrt{\frac{\Sigma_{residuals}}{\lambda + n_{residuals}}} \tag{2.17}$$

where λ is the L_2 regularization term, also known as the Mean Square Error.
By using the similarity score, the information gain is calculated. The information gain measures the quality of a tree structure. Particularly, which features or split-point maximize the gain. Mathematically, expressed as:

$$\mathcal{L}_{gain} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L}(g_i)^2)}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R}(g_i)^2)}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I}(g_i)^2)}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{2.18}$$

where the g terms are the sum of the gradient of the loss function, i.e the squared sum of the residuals. The sum of the denominator gives H, which is the hessian of the loss function. This corresponds to the number of residuals. The first term is the information gathered from the left child node, the second term from the right node, and the last one from the root node. γ is a hyperparameter, and represents a fixed threshold for the gain-improvement to keep a split [31]. Therefore the equation can be expressed as:

$$\mathcal{L}_{gain} = Left_{child-leaf-similarity} + Right_{child-leaf-similarity} - Root_{node-similarity}. \tag{2.19}$$

During the tree-building process, the algorithm stops for two reasons. The first occurs when the tree has reached its maximum predefined depth, whereas the second takes place if all the sum of the Hessian falls in one single node. In addition, at the end of the building process, the pruning step takes place [3]. Some nodes can be pruned if their gain value is lower than γ. If instead, the gain value is higher than γ then the pruning process stops and does not check the parent nodes of that particular branch.
Lastly, the last steps involve predicting the residual values of the trees and scaling these by the learning rate.

[3]Pruning decreases the size of the trees. In the process every split is evaluated. If given condition are satisfied, the split is considered redundant and removed.

2.2.6 Error Metrics

To evaluate the performance of the different methods, different error metrics are used. They relate the prediction of the method to the target value in order to improve the accuracy and the performance of the model.

1. **Mean Square Error (MSE)**
   Also known as L_2 regularization. It is defined as the sum of the squared difference of the residuals. It is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{N}(y_{true} - \hat{y}_{predicted})^2 \tag{2.20}$$

   where n is the number of samples. It is expressed in mm^2.
2. **Root Mean Square Error (RMSE)**
   It is the root of MSE. It is useful since the metric is expressed in mm,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{N}(y_{true} - \hat{y}_{predicted})^2} \tag{2.21}$$

3. **Mean Absolute Error (MAE)**
   Also known as L_1 regularization. It computes the expected value of the absolute error loss.

$$MAE = \frac{1}{n_{samples}}(\sum_{i=1}^{N}|y_{true} - \hat{y}_{predicted}|) \tag{2.22}$$

3
Methods

Firstly, an overview of the models and their features are described to better understand the workflow and the reason for some choices. Then, an overview of the workflow is presented. This includes data acquisition, extraction, and arrangement for dimensionality reduction methods to build the meta-model. The methods compared were Graph Autoencoder, Convolutional Neural Network, and Principal Component Analysis. Supervised regression tasks - introduced in Section 2.2.5, such as Random Forest, Gradient Boosting, and XGBoost - have been implemented on reduced dimensional space to link the simulation parameters to the latent space. Lastly, from the surrogate model, the latent space for the test data was reconstructed back to its original size to compare it with the original data and to estimate the error.

At the beginning of this project, a simplified model has been used to study and implement the previously mentioned workflow. The reason for this choice is that a simpler simulation is faster to run and analyze. In the simulation, a beam object moving into the 3D space has been modeled. The beam starts from an equilibrium position (see Figure 3.2a) and is first exposed to a force on one side (see the arrow in Figure 3.2b) along with another one that will obstruct the object to move, labeled as release force. At a certain point, the latter is removed and the object is free to move in the space . Since initially the object was blocked on one side, when released, it will move into the space with high-frequency oscillations. Geometrically, the beam is meshed by quads elements[1], shown in Figure 3.1. It contains 104 nodes. The simulation outputs 102 discrete timesteps which correspond to the duration 100 ms.

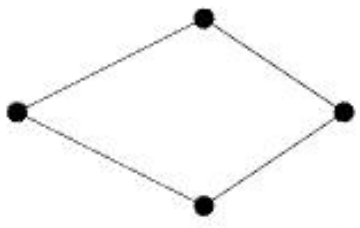
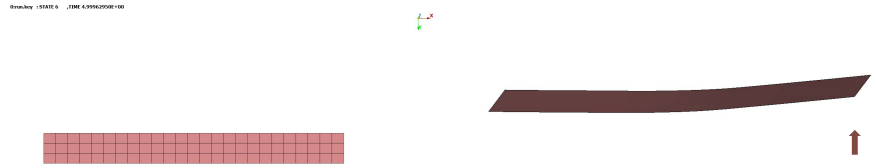


Figure 3.1: LS-DYNA Element shell

In the simulation, multiple parameters regulate the forces, the components setup, or the dynamics of the parts during the events. These will be referred to as FE simulation parameters. The value of the parameters has been chosen arbitrarily

[1] 4 vertexes per element



(a) Initial position.

(b) After 70ms, the constraint is removed and the object moves freely into 3D space

Figure 3.2: The beam model

following the empirical judgment. For the beam model, the FE parameters were the acceleration applied to the extremity of the beam and the release force and were varied as shown in Table 3.1.

FE Beam model			
Parameters	Unit	min	max
Release	[N]	1	15
Acceleration	$[m/s^2]$	0.1	1.2

Table 3.1: FE simulation parameters for beam object

Once the methods have been implemented and the first results obtained, the pipeline developed has been applied to the pre-crash phase simulations of a dummy model. The pre-crash simulation is composed of a Hybrid III dummy model of a 50th percentile man seated in the driver´s position inside the vehicle cockpit. The portion of the cockpit in the simulation contains only the driver's side, as shown in Figure 3.3a. The simulation starts with the driver being positioned in the seat, also known as the settling phase, of 300 ms. Afterward, a pulse is applied and the vehicle begins to move at a constant velocity, approaching a pre-crash phase, up until a break or steer takes place. The pre-crash phase lasts 2.3 seconds. The simulation is controlled by 5 FE parameters (see Table 3.1). The pre-pretensioner tightens the seat belt and makes it adhere to the dummy. A tense seatbelt limits the possibility of the shoulder belt to slide out in case of critical events. Different scaling factors have been chosen (see Table 3.2) and will influence the kinematics of the dummy during the pre-crash or in-crash phase. At 640 ms, the retractor of the shoulder belt and the lap belt are locked. The retractor forces the belt to lock in case of a sudden change in the delta velocity. The delay in the breaking of steering acceleration was varied between 0 ms, corresponding with the end of the settling phase, up to 700 ms, i.e. 1 second after the start of the simulation. This parameter changes the initiation of the pre-crash phase, where the maneuvers such as steering or breaking occur. The acceleration value of the two parameters changes the range of motion during the pre-crash phase. A higher breaking acceleration will increase the forward motion of the dummy. For every simulation, only breaking or steering are applied. This was chosen to limit the complexity of the simulation and thus aim at lower training time for the NN models.

FE Dummy model			
Parameters	Unit	min	max
Scaling factor Pre-pretensioner Force	-	0.001	2
Breaking or Steering Initiation [2]	-	0	1
Delay Breaking or Steering Initiation	[ms]	0.	700
Breaking acceleration in g	$[m/s^2]$	0.05	0.7
Steering Acceleration in g	$[m/s^2]$	0.05	0.7
Lock time belt - Pretensioner	[ms]	300	300

Table 3.2: Pre-crash FE parameters

(a) Initial position.

(b) After 600ms, dummy subjected to steering acceleration

Figure 3.3: Pre-crash simulation

The node history of the dummy is sampled every 25 milliseconds for a total of 81 timesteps. For the purpose of the project, only the nodes belonging to the dummy were taken into consideration, leaving out all the ones belonging to the vehicle (including the seat belt). The dummy model is composed of 7444 nodes. As for the geometrical structure, this is more complex than the beam simulation. It is composed of solid, shell, discrete, beam elements (see Theory 2.1).

3.1 Data extraction

As mentioned above, two different models have been used. The simulations were performed using LS-DYNA solver R11.1.0 for the beam model and R.9.3.0 for the pre-crash simulations.
To train the ML methods, multiple simulations with varying parameter values are needed. For this purpose, the Dynakit python library [3] was utilized. The parameters are obtained using a uniform distribution and random sampling. After obtaining multiple simulations, the FE simulation parameters were extracted from the d3hsp file, an output file from LS-DYNA solver.

[2]The breaking or steering initiation range is within [0,1], where 0 corresponds to only steering and 1 only breaking.

[3]Available at https://github.com/dynapy/dynakit

The data has been extracted from D3Plot of the simulation using the lasso-python open-source library from LASSO GmbH [34]. The nodal history acquired from this step represents the dataset used in the project. The D3Plot stores the time history of every node in the simulation object. Since the focus of this project is centered around the kinematics of beam and the dummy model, only nodes belonging to these were extracted. For the pre-crash simulation, since the dummy and the vehicle are moving at a constant velocity, a new reference point belonging to the dummy was selected and centered to form a new coordinate axis. If this step would have not been taken, the relative displacement of the nodes for the pre-crash adjustments of the dummy would have been obscured by the distance covered at a constant velocity. In both cases, the relative displacement was selected. Specifically, the initial position of the node was subtracted at every step to get the relative values. In both models, the displacement is measured in millimeters.
To sum up, from the D3plot the node history has been extracted and stored in 4D tensor of shape:

$$[n_{simulation}, n_{timesteps}, n_{nodes}, xyz_{displacement}] \tag{3.1}$$

and the simulation parameters stored in multidimensional tensor of size:

$$[n_{simulation}, n_{FE-parameters}] \tag{3.2}$$

80% of the displacement and parameters tensors were kept for training and 20% for validation.

3.2 Dimensionality Reduction I - GNN

The displacement tensor extracted from D3plot was structured in a 4D tensor of the following dimensions:

$$[n_{simulation}, n_{timesteps}, n_{nodes}, xyz_{displacement}] \tag{3.3}$$

The following approach was taken:

- the number of simulation and timesteps were stacked together to create 1D array of shape $n_{samples} = [n_{simulation} * n_{timesteps}]$. This results in having one sample equivalent to one time-step.
- the nodes and their displacement were extracted and to be later saved in the node features.

For this part of the project PYG - Pytorch Geometric Library was used [21]. As required by the library, the nodes' labels were relabeled from a range of [1,205] to [1,104] for the beam model and [1000001, 1011769] to [1,7444] for the pre-crash simulation.
In addition, in the pre-crash simulation, 3 nodes used for positioning were removed. These nodes were polluting the dataset of relative displacements, since they were moving at a constant velocity, they had a high displacement value. They were used only to position the dummy during the settling phase, and therefore they did not alter the final results. Therefore, the model ended up being composed of only 7441 nodes.

Lastly, one more step was taken to create the dataset. Since the GNNs take into consideration also the structure of the model, this needed to be extracted from the FE model definition. As mentioned in the Theory, the node belonging to solid, beam, discrete, and shell elements were extracted.
For every connection of the node, an adjacency list has been created. Particularly, not only the directed adjacent nodes have been inserted, but all the elements connected 1 unit from the selected node. To give an example, In Figure 3.4, an adjacency list for node 1 on imaginary squared structure is shown.
By following this criterium, the adjacency list will result in:

node 1 : $\rightarrow$ [2,3,4,5,6,7,8,9]

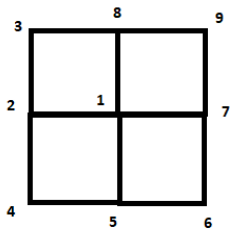


Figure 3.4: The edge list connections for an imaginary structure.

In fact, if we were to pull node 3, the entire structure would be deformed and not only nodes 2 and 8 would be influenced. Therefore, the choice of adding also node 1. After following this principle, the number of resultant edges is presented in Table 3.3[4].

	Number of edges
Beam model	[2, 656]
Pre-crash simulation	[2, 96 928]

Table 3.3: Number of edges in the models

Nextly, once data preparation was completed, a new dataset needed to be built, by joining the nodal history and the edge information. The Python Geometric library offers a class, named Data, that builds a geometric dataset. This class enables to define a graph following the definition given in 2.9, namely defining a node history and connection together.

$$data = Data(x = x, edge\_index = edge\_index) \tag{3.4}$$

The Data.x is the features tensor. It is $[n_{nodes}, 3]$ long, where n_nodes correspond to the label and the three dimensions are the x,y,z displacement for the specific node. Whereas the $Data.edge\_index$ stores the edge connections of the entire model.
In the graph convolution, the number of simulations and timesteps are stacked together in one dimension. As a result, one graph for every timestep is created. This means that the geometric dataset is a list of data classes long $[n_{sim} * n_{timesteps}]$.

[4]The first dimension 2 indicates the two edges connected to each other. See the adjacent list of Figure 2.8 in the Theory.

In addition, data has been normalized and standardized. In the first case, all the values have been scaled by the max increment of two timesteps. The scale factor was 13.13 mm for the beam model and 25.52 mm for the pre-crash simulation. The other method, performed only for the pre-crash simulation, was the z_score. It centers the dataset around the mean value and scales it for the standard deviation, as shown in equation 2.10.
The model and the features of an autoencoder have been introduced in section 2.2.4. Different architectures have been tested and studied, by changing different combination of layers size and depths of the network. The reason why the architecture between the two model are different lies in the fact that they are working with different dataset size, yielding to slightly different approaches and architectures. For example, the linear layer size in the pre-crash simulation was combined with a pooling due to high memory and RAM usage. Linear layers, due to their structure, take a considerable amount of RAM, having to store weight and value for every node. Specifically, the architecture adopted is shown in the the tables 3.4, 3.5.

Beam model			
Encoder			
Layers	In-channel	Out-channel	Output shape
Linear	3	3	[104,3]
GCN convolution	3	3	[104,3]
ReLU	-	-	-
GCN convolution	3	3	[104,3]
ReLU	-	-	-
Flatten	-	-	312
Linear	312	150	150
ReLU	-	-	-
Linear	150	12	**12** [5]
Decoder			
Linear	12	150	150
ReLU	-	-	-
Linear	150	312	312
ReLU	-	-	-
Reshape	-	-	[104,3]
GCN convolution	3	3	[104,3]
ReLU	-	-	-
GCN convolution	3	3	[104,3]
ReLU	-	-	-
Linear	3	3	[104,3]

Table 3.4: Graph Autoencoder Architecture for the beam model

Different setups have been implemented and will be presented in the Result section. A study for the latent space size has been performed to investigate how the performance of the model is influenced. For the beam model, the number of samples and the latent space size is varied and the performance of the model is evaluated.

Pre-Crash simulation			
Encoder			
Layers	In-channel	Out-channel	Output shape
Linear	3	3	[7441,3]
ReLU	-	-	-
GCN convolution	3	3	[7441,3]
ReLU	-	-	-
GCN convolution	3	3	[7441,3]
ReLU	-	-	-
Flatten	-	-	22323
MaxPooling	-	In channel /2	11161
Linear	11161	20	20
Decoder			
Linear	20	11161	11161
Upsample		2 * In channel	22323
ReLU	-	-	-
Reshape	-	-	[7441,3]
GCN convolution	3	3	[7441,3]
ReLU	-	-	-
GCN convolution	3	3	[7441,3]
ReLU	-	-	-
Linear	3	3	[7441,3]

Table 3.5: Graph Autoencoder Architecture for the Pre-crash simulation

The sample size considered and analyzed was 25, 50, 75, 100 simulations. For the pre-crash simulation, the comparison of sample size was not performed for computational limits, as the number of samples was 75 and the number of parameters varied was 5. In addition, different methods such as scaling and standardization have been tested.
The network was trained using Google Colab PRO Tesla T4 GPU. The models were trained with Adam optimizer and the hyperparameters are shown in Table 3.6.

Hyperparameters	Beam model	Pre-crash Simulation
Learning rate	1^{-5}	0.001
Epochs	100	500

Table 3.6: Training hyperparameters: epochs and learning rate

After the training, for the different implementation, the dataset was re-scaled back to the initial values to estimate the real error. MAE and MSE were computed to evaluate the result of the autoencoder.

3.3 Dimensionality Reduction II - CNN

The aim of developing a Convolutional Neural Network model lies in investigating the advantage of using geometrical information of the model to achieve the similar outcomes.
The architecture was chosen to be as similar as possible to the GNN autoencoder, but instead of graph convolution, 2D convolution was used. To investigate the difference in performance, the receptive field of convolution resulted in being similar to GNN. In Table 3.5, the number of layer of graph convolution used is two. Therefore it corresponds to aggregating and convoluting nodes up to 2 node distance. In this architecture to achieve a similar result, a 2x2 kernel was used. The CNN does not store the geometry of the model. Hence, by having two consecutive nodes does not mean they are adjacent in the FE structure. The approach to design the architecture was inspired by the image-pixel applications. The x,y,z displacement represent the channels of the input layer, similar to the color of RGB channel in images. The data arrangement was slightly different. The input data of the network was represented by a simulation instead of a singular time sample. In convolution, the number of nodes and the time dimension represent the width and the height of the input window. Therefore, the input batch is of the following shape:

$$[n_{simulation},\ xyz_{disp},\ n_{nodes},\ n_{timesteps}] \tag{3.5}$$

The architecture used was the following shown in Table 3.7:
Some of the layers have already been introduced in the first architecture. However, there are two new layers, convolution2D (see Section 2.2.2, Figure 2.9) and MaxPool2D (see Section 2.2.4) .
The convolution 2D works over a 4D input [N, C, H, W], where N is the batch size, C denotes the number of channels, H and W are the height and width of input

Pre-Crash simulation CNN				
Encoder				
Layers	In-channel	Out-channel	Kernel	Output shape
Linear	81	81		[1,3,7441,81]
ReLU	-	-	-	[1,3,7441,81]
2D convolution	3	3	[2,2]	[1,3,7440,80]
ReLU	-	-	-	[1,3,7440,80]
2D convolution	3	3	[2,2]	[1,3,7439,79]
Max Pooling 2D		scale-factor 4	-	[1,3,1859,19]
Flatten	-	-	-	[1,105963]
Linear	105963	20	-	[1,20]
Decoder				
Linear	20	105963	-	[1,105963]
ReLU	-	-	-	[1,105963]
Reshape	-	-	-	[1,3,1859,19]
Upsample	-	[7439,79]	-	[1,3,7439,79]
2D convolution	3	3	[2,2]	[1,3,7440,80]
ReLU	-	-	-	[1,3,7440,80]
2D convolution	3	3	[2,2]	[1,3,7441,81]
ReLU	-	-	-	[1,3,7441,81]
Linear	81	81		[1,3,7441,81]

Table 3.7: CNN Architecture for the Pre-crash simulation

planes. In this case, to maintain the same methodology, the batch size has been kept to one as in the GNN architecture. This indicates that the model is trained with one sample at the time. As already mentioned, the number of channels correspond to three, for the x,y,z, displacement. The convolution kernel slides along the window formed by the width and height of the input window. For every convolution, the size of the input size will shrink. The final output is given by the formula:

$$H_{out} = H_{in} - (kernel_{size} - 1) \tag{3.6}$$

$$W_{out} = W_{in} - (kernel_{size} - 1) \tag{3.7}$$

The MaxPool2D downsamples - by a factor of 4 - the input plane by extracting the max features over the kernel size. The output of the max pool is given by the same formula of the convolution, presented in 3.7.

3.4 Dimensionality Reduction III - PCA

As mentioned in Section 2.2.4, the dataset obtained from the D3plot has been standardized using z-score method.
The data needed to be arranged in a way suitable for the PCA method on sklearn python library [35]. The PCA function implemented works with input shape of

$$n_{samples}, n_{features} \tag{3.8}$$

From the 4D tensor, the $n_{simulation}$ and $n_{timesteps}$ along with n_{nodes} and $x, y, z_{displacement}$ were stacked together to form a 2D tensor. By shaping the input dataset in the 2D tensor dimension, the next step was to decide the number of components or explained variance. The percentage of explained variance chosen was 99% percent from the dummy model, resulting in 5 components. After inputing these data in the PCA function the output of the reduced dimensional space was of shape:

$$[n_{samples}, n_{components}] \tag{3.9}$$

The DR in PCA consists of reducing the number of features to the number of components, in this particular case to reduce the node history for every sample to a few components.
To compute the error from the PCA, the inverse of PCA transformation to input shape was computed and MSE and MAE were calculated for the entire dataset.

3.5 Regression

Both PCA and GNN are using the same data arrangement for regression. Therefore will be treated once for both of the methods.
After obtaining the latent space for the model, the next step consists of linking this one to the FE parameters. The regression takes as input FE parameters and will relate the parameters to the target, the latent space.
The FE parameters, after being extracted from the d3hsp - an output file from LS-DYNA - have been stored a tensor of shape

$$[n_{sims}, n_{parameters}] \tag{3.10}$$

Therefore it needed also to be arranged to match the same shape of the latent space. Two steps were taken towards this setup:

- The first dimension was brought to match the one of the reduced dimension, therefore $n\_sim * timesteps$
- Due to the arrangement simulation * timestep, the result obtained is one low-dimensional space for every timestep. Therefore, time becomes one additional parameter to connect k-latent space with k-timestep.

To give an example, given α, β, two FE parameters for a general instance, the new tensor for the FE is arranged as shown in Table 3.8.
Three algorithms were used: Random Forest, Gradient Boosting, and XGboost. The first two are from the sklearn library [35] and the last one from XGBooost library [31]. For all the regression methods, the hyperparameters were tuned using two different optimization frameworks, the RandomizeSearchCV from Sklearn [35] and Optuna library [36].
The RandomSearch performs a Random Search of some fixed parameters set where every setting is chosen over a distribution. It also allows to perform KFold cross-validation. The number of folds for the KFold cross-validation was chosen to be 5, i.e. the training dataset was divided in 5 folds and 80% was used for training and 20% for validation. The best set of parameters is chosen by using a fit and score method.

Simulation	Input	Target
1	α_1, β_1 ,t_1	$Latent\ space_{t1}$
	α_1, β_1,t_2	$Latent\ space_{t2}$
	..	..
	α_1, β_1 ,t_k	$Latent\ space_k$
n-1	$\alpha_{n-1}, \beta_{n-1}$,$t_1$	$Latent\ space_{t1}$
	$\alpha_{n-1}, \beta_{n-1}$,$t_2$	$Latent\ space_{t2}$
	..	..
	$\alpha_{n-1}, \beta_{n-1}$,$t_k$	$Latent\ space_k$
n	α_n, β_n,t_1	$Latent\ space_{t1}$
	α_n, β_n,t_2	$Latent\ space_{t2}$
	..	..
	α_n, β_n,t_k	$Latent\ space_k$

Table 3.8: Regression: FE simulation parameters and Latent space

Optuna uses a Bayesian approach to sample the hyperparameters. In contrast to the random search where the parameters are sampled from a distribution, the choice for the set of parameters is based on previous evaluations. For simplicity, it can be summed up as two-step process, the study and the trial. First, a *study* is initiated to evaluate the objective function. In the study, Optuna takes in consideration a subset of the hyper-parameters given. After initiating a study, the *trial* takes place. In this part of the process, the algorithm evaluates the set of parameters by giving a score. After testing and evaluating for n trials, the framework output the best set of parameters with the best score.

After finding the best set of parameters for the regression methods, the regressors were fitted to the training set. Lastly, the validation set was predicted. For the PCA, the inverse transform of the PCA is used. It restores the number of feature for every sample and scales it back. For the GNN instead, the output from the regression is decoded back by using the trained decoder. Next, the MAE and MSE are computed for original test set.

3.6 Animation on META post-processor

The last part of the pipeline consists of recovering the predicted data and arranging them to animate the reconstructed simulation in META post-processor. To goal is to have an animation of the model. To accomplish this last part of the pipeline, a script from Karl-Johan Larsson (PhD from Chalmers) was used. The script enables the META post-processor to read LS-DYNA format files and build the animation from a static base model. To be able to achieve this, the script was slightly modified to read a multi-step simulation. In addition, the simulation needed to be written in LS-DYNA format. To implement this, every timestep of the simulation was created as a unique file. In the file, the total displacement for every node was written. The total displacement was obtained by summing the initial position to the relative

displacement output from the surrogate models, separately for x,y,z-displacements. And in the case of the GNN the reference coordinate system was restored to the initial one.

4
Results

The performance of the different models is compared and presented in this chapter. The chapter is divided into two sections, the first regarding the beam model and the second regarding the pre-crash simulation. For both models, the dimensionality reduction will be presented first, followed by the regression methods.

4.1 Beam model

4.1.1 Dimensionality Reduction

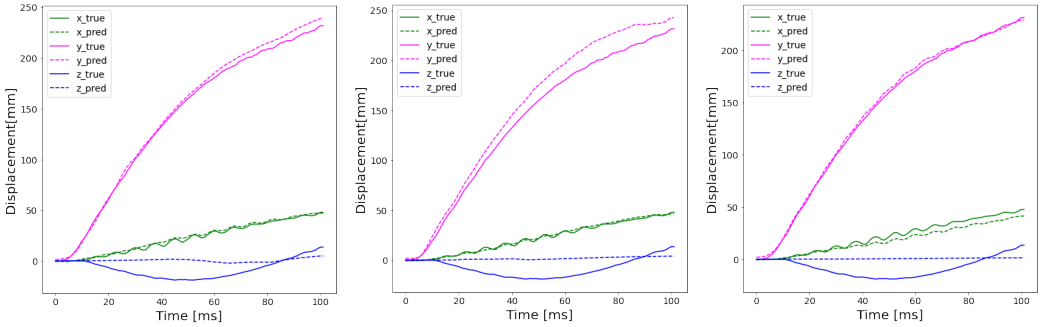
GNN

The structure of the Graph autoencoder is presented in Table 3.4. In the Table 4.1 below, the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) on the train and test data is shown for data. The latent space dimension tested were 10,25, 50. The size was arbitrarily chosen. The error in both the training and testing set reduces when the latent space is increased. By having all the same epochs and learning rate, the only parameter changing is the size of the linear layer, which can be held responsible for this outcome. A reduced latent space can reduce the capacity of the layer to learn throughout the training. In fact, a latent space size belongs to one of the hyperparameters that need to be optimized in an architecture.

Latent space size		Train	Test
10	RMSE MAE	7.85 5.06	9.78 6.373
25	RMSE MAE	7.82 5.03	9.14 5.763
50	RMSE MAE	6.83 4.41	8.49 5.07

Table 4.1: MAE and RMSE for different latent space size

In the following Figures 4.1, node 99 - located on the upper edge of the beam - trajectories are displayed. The trajectory of x,y,z is plotted as function of time. In Figure 4.1c, the prediction is closer to the true values rather than 4.1b, 4.1a. Moreover, it seems to better interpolate high displacement of the y axis, whereas the high frequencies deflections are better captured by 4.1a.



(a) Latent space of 10 **(b)** Latent space of 12 **(c)** Latent space of 50

Figure 4.1: GNN evaluation on Node 99 with respect to different latent space sizes.

In addition, a comparison of the test and train error has been made when given a fixed latent size, learning rate, and epochs. Figure 4.2 shows MAE for a size of 25, 50, 75, 100 samples. The error decreases when the sample size is increased. The maximum error of 21.552 mm of the test set for 25 samples is caused by an unrepresentative test dataset of only 5 samples. In this case, there are enough samples for the training set, but the model struggles to generalize the predictions for the test set. By increasing the number of samples to 50, this issue is overcome and the model reaches a good fit. From 50 up to 100 samples, the model seems to have found a plateau, that even when increasing the number of samples it does not yield evident differences.

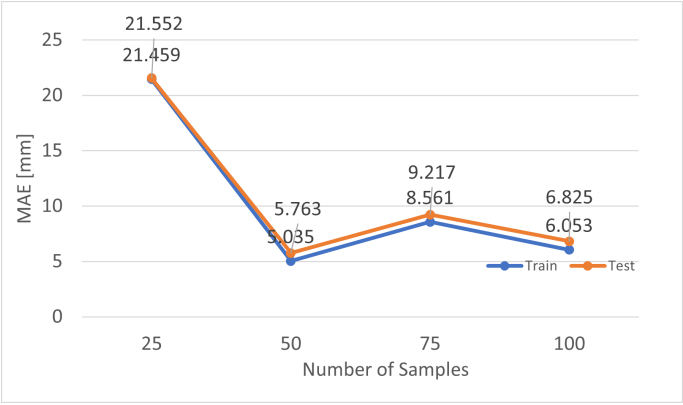


Figure 4.2: GNN Train and Test MAE for different sample sizes

PCA

Different numbers of components capture different percentages of variance in the dataset. In Figure 4.3, this relationship is highlighted. Interestingly, the 3 components capture 90% of the variance. Whereas going from 99% to 99.99% the number of components changes from 7 to 13.

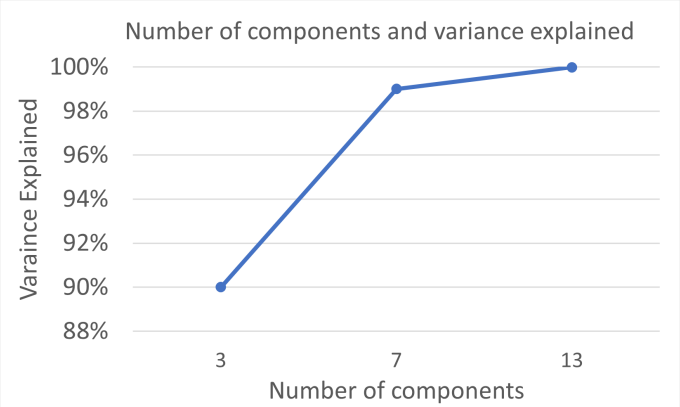


Figure 4.3: PCA - explained variance with respect to the number of components for the beam model

In Table 4.2 and Figure 4.4, the number of components and the number of variance explained by each component is presented. As introduced in Section 2.2.4, the first component captures the majority of the variance. It is also possible to see that from the 8th component the total amount of variance captured is 1 %, with a contribution of 0.41 % from the 9th component.

N. of components	Total Variance
3	90%
7	99%
13	99.99%

Table 4.2: Number of components and explained variance

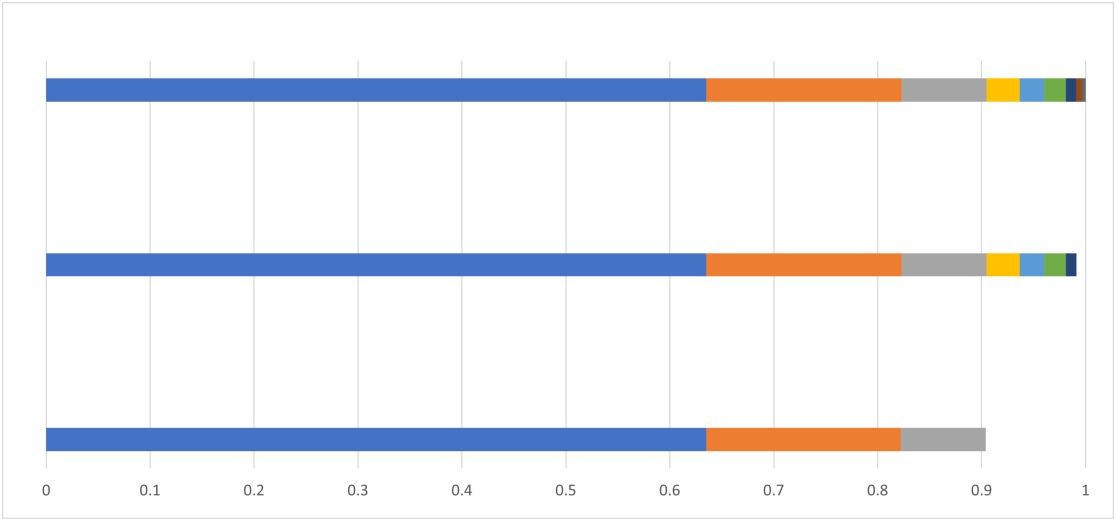


Figure 4.4: Number of components and explained variance

Figure 4.5, shows the relation between number of components and MAE on the training dataset of the method. Capturing more variance, and therefore having

more components, results in lower training error. Moreover, the graph shows also the same trend for different samples sizes of 25, 50, 75, 100. The graph starts at 7.84 for 100 samples, 7.8 for for 75 samples, 7.26 for 50 samples and 7.037 for 25 samples. The minimum error reached for 13 components is 0.0023 by samples size of 50. Analogous results are achieved by the other samples sizes, slightly different values after the third decimal.

As shown in Figure 4.5, different number of components yields different training error. In Figures 4.6, node 44 - located on the lower edge of the beam - of the training set is plotted for different amount of components. Clearly, in Figure 4.6a, the prediction is inaccurate from the time 0, and pronounced particularly for the z axis. On the contrary, when using 13 components, the predicted values and the true ones are overlapping by being able to capture the high-frequency deflection around 0 of the y axis.

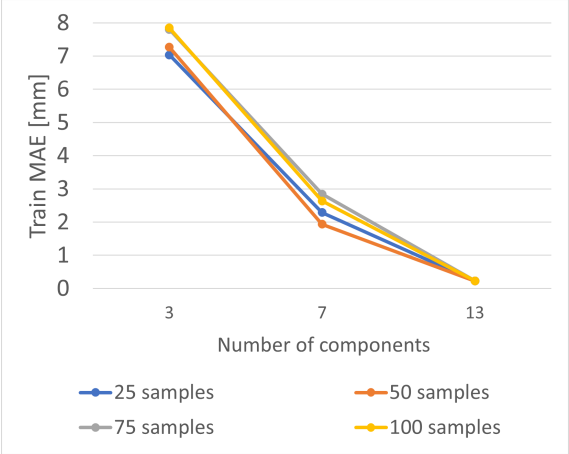
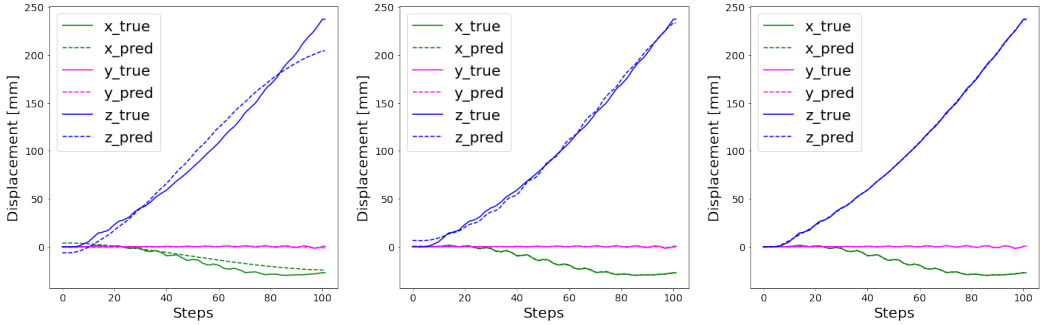


Figure 4.5: PCA beam model: Train MAE error with respect to the number of components

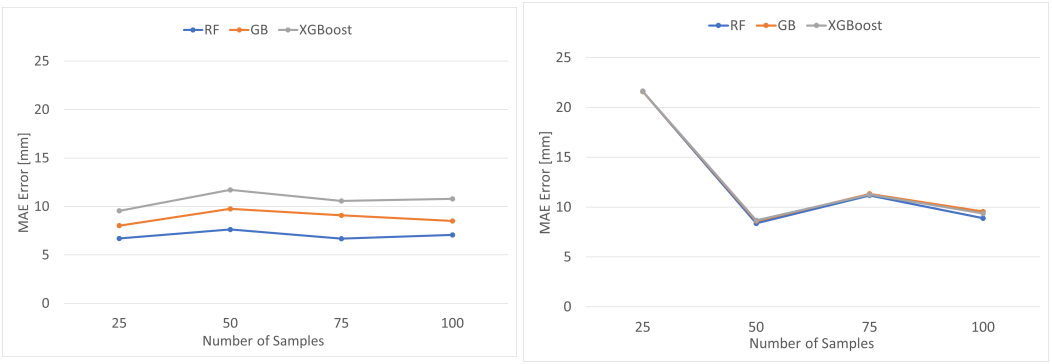


(a) 3 components **(b)** 7 components **(c)** 13 components

Figure 4.6: Beam model: node 44 with respect to the different number of components.

4.1.2 Regression

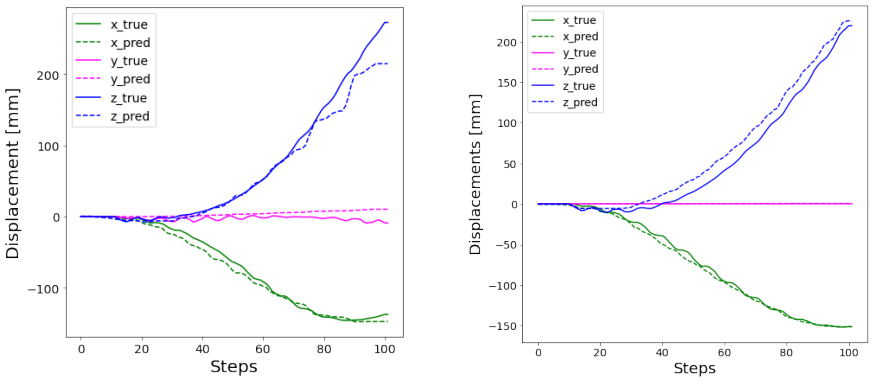
For the beam model, in Figure 4.7, a comparison between the number of samples, regression methods, and validation error is presented for the Graph Autoencoder and PCA. The latent size chosen was 7 for the PCA and 25 for GNN. The error plotted is the average MAE of the validation set.



(a) PCA-regression **(b)** GNN - regression

Figure 4.7: Beam model: final MAE with respect to sample size.

While on the autoencoder the decreasing trend is clearly visible, the same is not for the PCA. However, it is possible to notice that there is a decline from 50 sample size. The minimum error is reached in PCA with the Random Forest algorithm. Lastly, in Figure 4.8, the final trajectory for both method have been plotted from a sample node.



(a) PCA- RF **(b)** GNN- RF

Figure 4.8: Beam model: node 99 after the RF regression

4.2 Pre-crash Simulation

In the section below the result for the pre-crash simulation are presented. The number of simulations used in the dataset is 75, representing all the simulations available.

4.2.1 Dimensionality Reduction

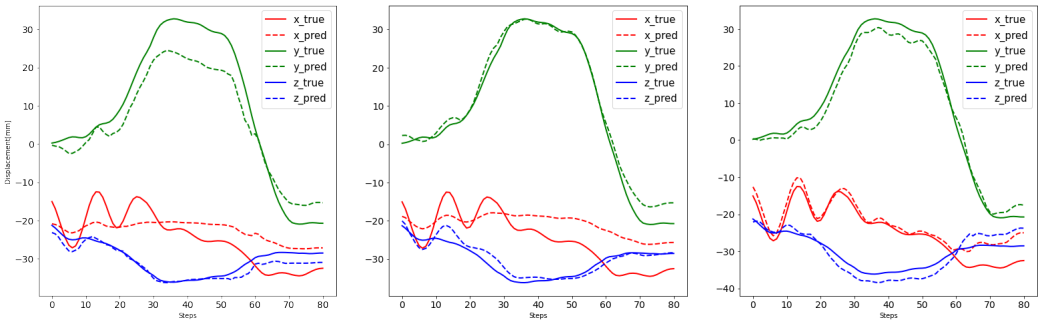
GNN

For the Pre-crash simulation, a slightly different architecture was used. The architecture is presented in Table 3.5.
Table 4.3 shows the train and test Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) for different latent space sizes. Specifically, the chosen sizes were 5, 25, and 100. It is possible to notice that the lowest error of 5.11 for the test set has been reached with a latent size of 100. Overall, Table 4.3 shows that a latent space of 5 or 100 are the achieved better results than 25.

Latent space size		Train	Test
5	RMSE	6.83	7.38
	MAE	4.47	5.174
25	RMSE	8.22	8.95
	MAE	5.48	6.01
100	RMSE	6.67	7.40
	MAE	4.63	5.11

Table 4.3: MAE and RMSE for different latent space size

The plots for the different latent sizes of training sample 1 for node 2004041 - located in the leg of the Dummy - are presented in Figure 4.9. In 4.9a the model reconstruction is far from the ground truth, especially for the y and x displacement. Of particular interest is the prediction of the x axis (in red) in Figure 4.9c. As it can be noted, in the first 20 timesteps, the model predictions reconstruction converge to almost overlap the ground truth. In Figure 4.9b y displacement in the dotted line is closer to the ground truth than in Figure 4.9c. However, Table 4.3 shows that the global MAE for the simulation is lower, indicating that the prediction for a single point may be worst but the general performance of the model improved.



(a) Latent size of 5 **(b)** Latent size of 25 **(c)** Latent size of 100

Figure 4.9: Pre-crash simulation - GNN: Node 2004014 for different latent sizes

Scaling vs. Standardization In Table 4.4, the train and test MAE for the different scaling method is shown. As it is possible to notice, the difference between

the two is clear. The standardization performs better than max-scaling. In the AppA, different nodes from the various part of the model have been plotted.
At the end of the training, the training and test error was computed. The results show as follows in Table 4.4.

	Max-time Scaling	Standardization
Train MAE [mm]	4.63	3.61
Test MAE [mm]	5.11	3.95

Table 4.4: MAE for max timesteps scaling and standardization

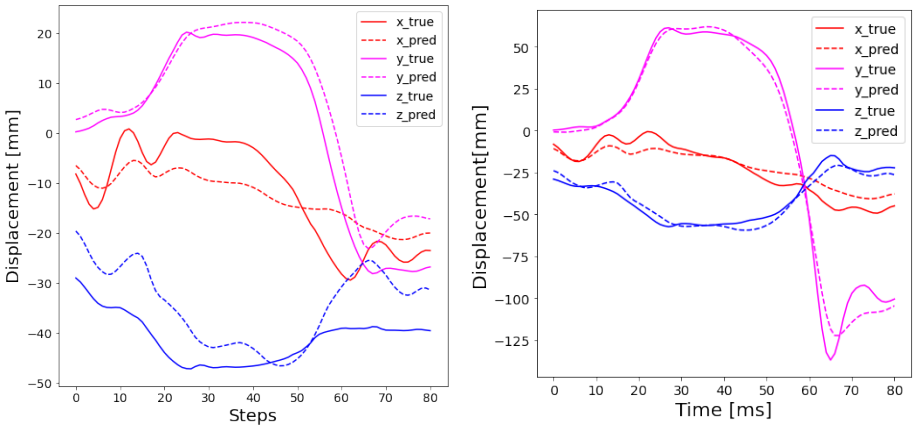
Convolutional Neural Network

The CNN architecture is presented in Table 3.7. The network was trained for 500 epochs and a learning rate of 0.001. The training and test error achieved are presented in the table 4.5:

	MAE [mm]
Train	5.044
Test	5.770

Table 4.5: Training and Test error for CNN architecture.

Node 2009276 - located in the right foot of the dummy - of the train and test set is plotted below in the Figures 4.10a, 4.10b below.



(a) Node 2009276 - Train set **(b)** Node 2009276 - Test set

Figure 4.10: CNN performance on training set - Node 2009276

PCA

Similarly to the beam model, the relationship between variance and the number of components is presented in Figure 4.11. In this case, the number of components to

capture 90% of the variance is 2. However, in the pre-crash simulation 26 components are needed to catch 99.99% of the variance.

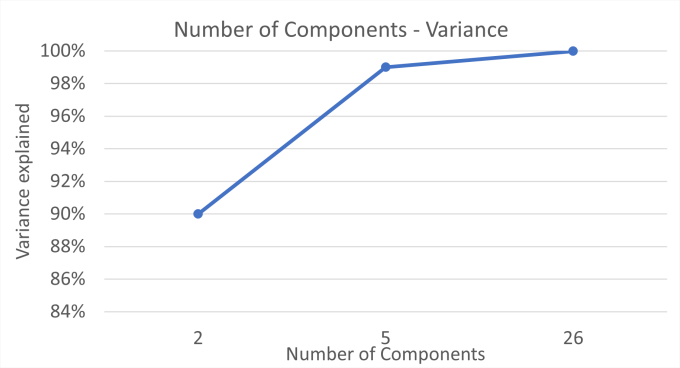


Figure 4.11: Pre-crash PCA: Explained variance vs. number of components.

Moreover, Table 4.6 and Figure 4.12 show the number of components along with the variance captured for each component. Interestingly, in this case, starting from the fourth component one percent of the variance is expressed. In the case of 26 components, the amount of variance carried from the last components is negligible.

N. of components	Total Variance
2	90%
5	99%
26	99.99%

Table 4.6: Amount of variance expresses by each component, increasing the number of components

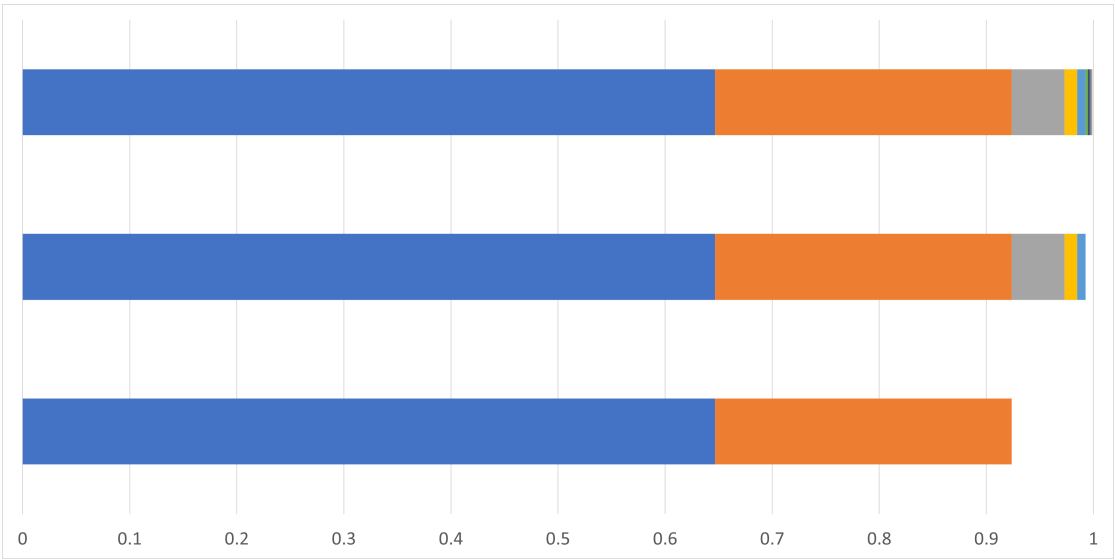
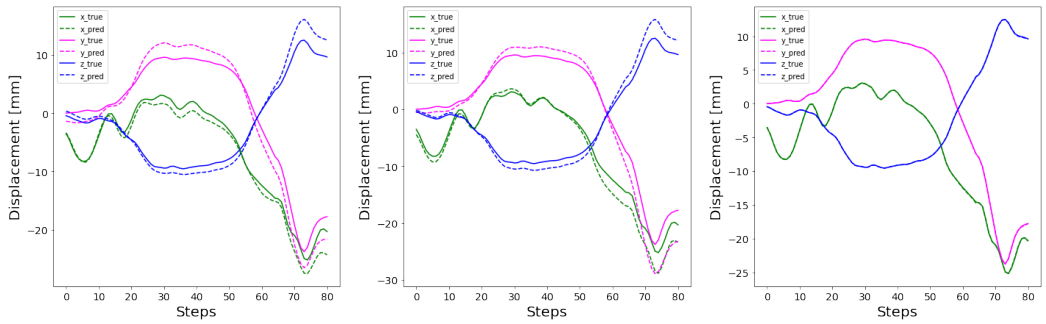


Figure 4.12: Amount of variance expresses by each component, increasing the number of components

Nextly, in Figure 4.13, node n. 2005770 - located in the right leg of the Dummy - from the training set is plotted in relation to the variance captured. It is interesting to notice, in Fig 4.13b that going from only 2 to 7 components increases the accuracy to almost having an overlap of true and predicted value. Whereas the difference between Figure 4.13b and Figure 4.13c is less pronounced.



(a) 90 % explained variance - 2

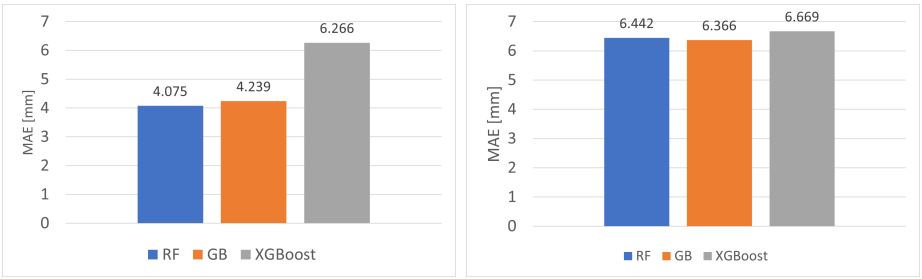
(b) 99 % - 7 components

(c) 99.99 % - 26 components

Figure 4.13: PCA - varying size of components - Node 2005770

4.2.2 Regression

In the Figures below 4.14, a comparison of the test set MAE for the three methods is presented. For the PCA the best method is the Random Forest, whereas for GNN is the Gradient Boosting. PCA paired with Random Forest achieves the lowest error of 4.075 mm. XGBoost has the highest error if paired with PCA. The same counts for the GNN with an error of 6.669 mm.



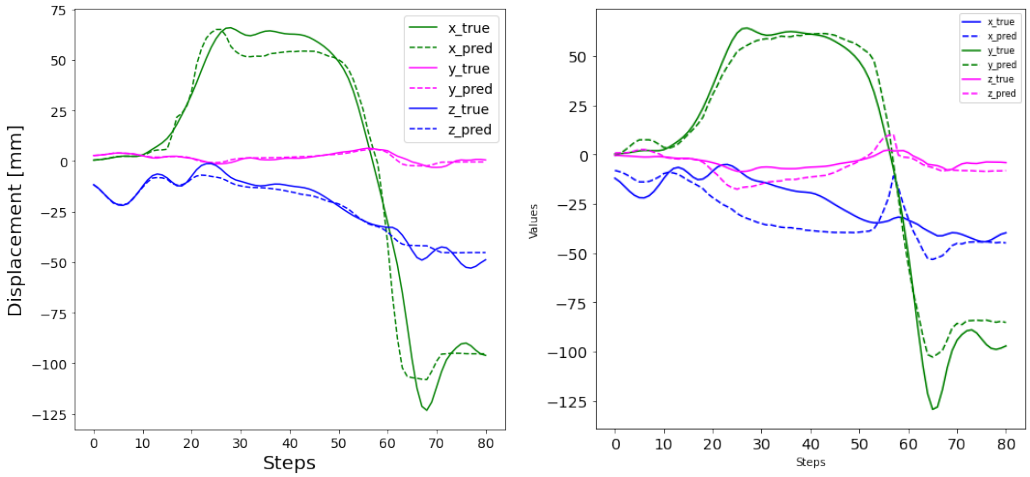
(a) PCA

(b) GNN

Figure 4.14: MAE for Random Forest, Gradient Boosting, and XGBoost in test set

In addition, node 2004014 - located in the left leg of the dummy - is plotted for both methods paired with GB in Figure 4.15. Overall, displacement for Figure 4.15a is closer to the ground truth than Figure 4.15b. For the y displacement (the magenta line), the reconstructions are similar. However, the behaviour differs for the x (the red line) and z displacements (the blue line). In Figure 4.15a, the prediction and ground truth are overlapping for the first 20 steps for the x displacement, indicating

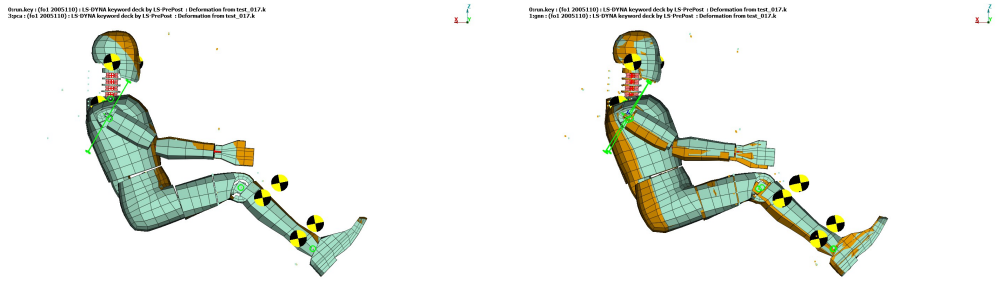
the good performance of the model in reconstructing the latent space. In contrast, in Figure 4.15b the reconstructed displacement is not converging to the ground truth as much as the other model, yielding an increase of the MAE error for the node plot.



(a) PCA-RF **(b)** GNN-GB

Figure 4.15: Pre-crash Regression: Node 2004014 from test set

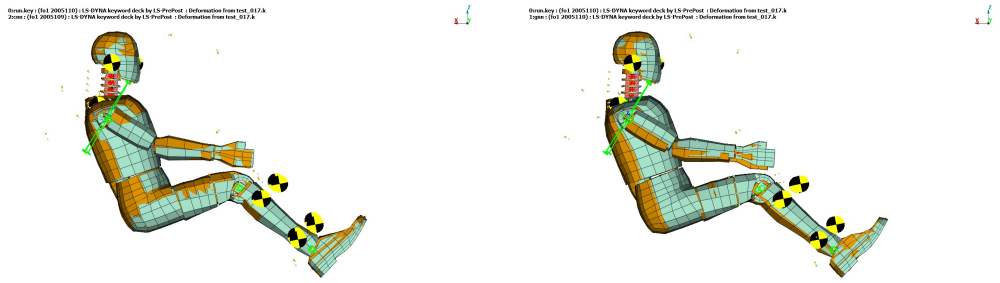
PCA vs GNN Lastly, in the following Figures 4.16 - A.7, a comparison of the animated simulations is presented. Both methods used gradient boosting regression. In green is the ground truth, which is the original simulation. In orange, the reconstructed simulation. The timesteps selected were the 17th and the sequence from the 25th to 29th, specifically time 729.99 and from 929 to 1029 ms. The sequence of frames was chosen to display the reconstruction in subsequent frames where the breaking of steering is occurring at its maximum. As shown in Figure 4.16, it is possible to notice that most of the reconstruction error is located in both the upper and lower limbs, the head, and the pelvis area. PCA achieved better results, coherently with what shown in Figure 4.14. Here, in this chapter, only the first frame is shown. For a better comparison, the other frames are shown in the Appendix A.



(a) PCA-GB **(b)** GNN-RF

Figure 4.16: PCA-GNN comparison: after 729 ms

CNN vs. GNN Additionally, another comparison is presented between the CNN-GB network and the GNN-GB. The same frames have been selected. At 729 ms, the two models have analogous results. In the frame sequence CNN has better performance and lower error in both the upper limbs and back area.



(a) CNN-GB **(b)** GNN-RF

Figure 4.17: CNN - GNN comparison: after 729 ms.

5
Discussion

The initial questions and goal of the project were to study and analyze in-depth the GNN potential. Differently from CNN, GNNs store and use geometrical and spatial information in the convolution, therefore have the potential of achieving better results. Particularly, the graph built by the adjacency matrix is a similar copy of the model FE mesh. Different methods of DR were implemented, evaluated and compared. The outcome and the performance of the models were determined by many factors. In the following sections, some of the most important aspects will be addressed.

5.1 Dataset

Starting from the dataset, for instance, different scaling techniques can be implemented. Normalization, scaling, and standardization are the most common. They all act on the dataset differently. The presence of outliers[1] can influence the entire workflow, and therefore different methods are adopted to minimize the effect of outliers.

For example, in the beam model, 10 isolated nodes were not moving throughout the simulation. These nodes were not belonging to the beam object but were used by the solver during the simulation. In the beginning, these nodes were contaminating the dataset and the network was terrible at predicting their trajectories. When the stationary nodes were removed from the beam object, the network achieved lower training and testing errors. Similarly, in the pre-crash simulation three nodes that were extracted from D3plot and used during the settling phase. The average relative displacement of the nodes in the dummy is 25mm. The relative displacements of the outliers were 20m (a thousand times more!). When normalizing or scaling, due to high values, the mean of the dataset was offset from the real and true mean, and undermining the loss function throughout the learning process. The solution adopted was to create a sub-graph that left out the outliers.
A comparison of different scaling methods was described in Section 4.2.1. The standardization showed to perform slightly better than the max scaling with a difference in the error of around 1 mm. This behavior can be explained by the fact that the polluting nodes were the only clear outliers and the relative displacements did not

[1]Outlier is a value that differs significantly from the rest of the dataset and interferes with the learning process. In ML, several approaches are taken to reduce the effect on the dataset in order not to affect the training quality.

differ noticeably from each other, which is logical since the nodes are moving together. As a result, from the very first step, every choice determines the outcome of a method and shows that data cleaning is a fundamental part of a ML pipeline. In addition, another factor influences the results. For the beam model, the sampling rate was 1000 Hz (one sample per ms). For the pre-crash simulation, the solver sampled every 5 ms. However, in the data extraction process two timesteps differed by 25 ms. These choices were the result of computational limitations. The lower the sampling, the higher the memory usage to store the dataset. Choosing a sampling rate of 5 ms was not feasible for all the simulations during this project. But increasing the sampling rate, and hence the timesteps, increase the number of samples for training. As shown, by the samples comparison of the beam model, the higher number of samples, the lower the error.

5.2 Architecture

The arrangement of the dataset determines the ML architecture. Choosing an input size of one timestep or one entire simulation affects the hyperparameters and the learning process of the network. Also, the hyperparameters such as learning rate, epoch, layers size, and batch size are of great importance and challenging to tune.

Batch size

In GNN, the choice of the batch size is fundamental. At the beginning of the project, a small CNN on the beam model was implemented to learn the Pytorch API. In this example, the hyperparameters were tuned accordingly to the case and architecture. In the GNN architecture, using the same batch size of 5, yielded the network not learning any feature in the training process. Therefore, hyperparameters are unique choices for each application that are time-consuming to tune manually.

Over-smoothing

Generally, for datasets such as the Hybrid III dummy, deeper NNs are used as they have more trainable parameters. A limitation for GNN was the so-called *over-smoothing*. This phenomenon is very common in the GNN field [44]. Over-smoothing occurs when there are several layers of GNN convolution. This leads to features of the nodes being convoluted many times. As a result, they become similar to each other due to several aggregations [44]. This phenomenon observed at the initial stage of the implementation limited the depth of the network and the solution adopted was to reduce the number of convolution layers in the network.

Limitation GNN

There is a rather well-known and successful architecture in other applications and this proved to achieve good results in the project from Gurram and Venkata [8]. This architecture is known as convolutional U-net. Its name comes from the shape of the

network. They are based on convolutional auto-encoders, namely an architecture alternating convolution and pooling layers. The embedded space is achieved both by the contribution of convolutional and pooling layers. Unfortunately in the GNN field, pooling layers are a challenging feature and are not yet fully established. The reason for such challenges derives from the graph shape and its lack of rigorous structure, unlike images. Therefore, when reducing the input features there are no fixed criteria to aggregate them. More importantly, no current implementation can unpool the graph to its original form. This is due to the information of the edges being lost when the graph is pooled which cannot be reconstructed afterwards. Moreover, the current limitation comes from the maximum number of nodes in a graph that can be process by the four most common pooling methods. According to Grattarola and al., this limit is 10000 for dense adjacency matrices and 30000 for sparse ones [38] [2]. This imposes a limitation that might be extended in future work. As a solution, a simpler architecture with no graph-pooling was chosen. This became one of the weaknesses of the U-nets architecture implemented in this project. The encoding power lies exclusively on the Linear and MaxPool1D layers. In fact, the pooling layer used is Maxpool1D on a one-dimensional tensor and no convolution was used after pooling.

5.3 Comparison of Dimensional Reduction

Given the two different datasets, the number of components needed to capture the same variance varies, as shown in Figure 4.3 and 4.11. For a smaller dataset, like the beam model, only 13 components are necessary to capture the 99.99% of variance of the dataset (see blue line in Figure 4.3). In contrast, the pre-crash simulation needs 26 components. To reach 90% of the variance, only three components for the beam model and five components for the pre-crash model are needed. This result confirms that a reasonable cut-off for PCA is three components as presented in [19] (see Section 2.2.4). While higher numbers of components can yield an improvement, it becomes more time consuming. That is because in the regression, each FE parameter needs to be mapped to each variable in the latent space.
As seen in the results, PCA proved to be the best method for Dimensionality Reduction by reaching the lowest error. The GNN has a higher MAE in the training and test dataset in both beam and pre-crash simulation due to the limitation of a missing pooling layer. Surprisingly, the CNN proved to be as good as the GNN with a latent space of 25.

5.4 Comparison of Regression

Of the three regression methods tested, the Random Forest achieved the best result in the testing in both PCA and Graph Auto-encoder for the pre-crash simulation. In contrast, XGBoost was the weakest regression method. The advantage of Random

[2] Dense matrix means the majority of entries in the matrix is non-zero.

Forest is having fewer parameters to tune. This makes tuning more efficient. On the contrary, XGBoost has the potential to outperform Gradient Boosting as it is an enhanced implementation of the latter. However, it is more challenging to tune due to the number of hyperparameters. Regarding this issue, hyperparameters optimization frameworks are helpful. The remaining challenge lies in the necessity to choose arbitrarily the range of values for each hyperparameter leading to a trade off between optimal parameters and time consumption (see Section 3.5). For the XGboost, a hyperparameters study needed to be performed to understand which parameters were of major influence in the training process.

5.5 From the dummy model to the HBM

It would be possible to replace the dummy model with an actual HBM model in the pre-crash simulation. The adaptation of the methods would be straight forward but the simulation would become computationally demanding. Particularly, creating training datasets with the FE solver would require time [3]. This was not feasible within this project, hence the usage of a simpler model. This issue could be resolved using more CPU cores and generate simulations in parallel. Working with the dataset would take more RAM for processing and using the data in the training process. For example, the dataset of the simple dummy model used for the pre-crash simulation had already a size of 2 gigabyte for 75 samples. The HBM dataset of 500 000 nodes would be considerably larger, about 60 times more. Challenges would arise when processing the data for the methods and during the training phase, especially in the DR part, as the regression would work with a reduced dataset. As the focus of this project was more on method development, a simpler model enabled a more thorough investigation of feasible methods.

5.6 Computational resources

In this section, a discussion and overview of the resources and time consumption of the methods is given.

Beam model

The number of simulations produced for the project was 100 samples. The amount of time for each simulation was of 95 seconds with 4 CPU cores. The overall time to produce 100 FE simulations was of 2.6 hours. The node displacement dataset stored occupies 12.1 MB.
In Figure 5.1 below, a comparison of the training time between the PCA and GNN is given.

[3] A single simulation with a HBM model requires 112 hours - 4.6 days - with 64 CPU cores. For the dummy only 4 CPU cores were used.

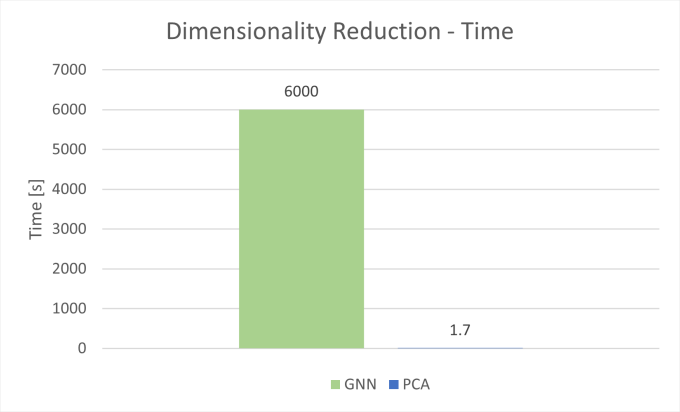


Figure 5.1: Time consumption of different Dimensionality reduction method

It is possible to notice that the PCA is the fastest with only 1.7 seconds. Whereas the for GNN, when trained for 100 epochs, it takes 3500 times more.

To give a comparison, let´s suppose to reproduce 500 simulations with LS-DYNA solver. The time required for the solver would be of 13 hours - or 47500 seconds. With Dimensionality Reduction, we could train the model with 100 simulations and predict the other 400. The FE simulation parameters would be given to the regression model and reconstructed back with the decoder or PCA inverse function to get a reconstructed HBM model. However, to compute the total time for the Dimensionality Reduction methods, we would also need to add the time to produce 100 simulation with solver. As shown in the table 5.1 the time to predict 20 simulations for GB is 0.4 seconds, for RF is 0.2 seconds and for XGBoost is 0.1.

	Train	Predict
Random Forest	0.1	0.2
Gradient boosting	10.4	0.4
XGBoost	22.9	0.1

Table 5.1: Beam model: Time required for different regression for training and prediction

If we would convert these times for 400 simulations, it would become 2 sec for GB, 4 for RF, and 8 for XGBoost. However, these time computations result negligible when compared to the time required from the solver to produce one single simulation. Therefore, they will not be taken into consideration, and only the different Dimensionality Reduction will be taken into account.

As a result, the time required for 500 simulations would be:

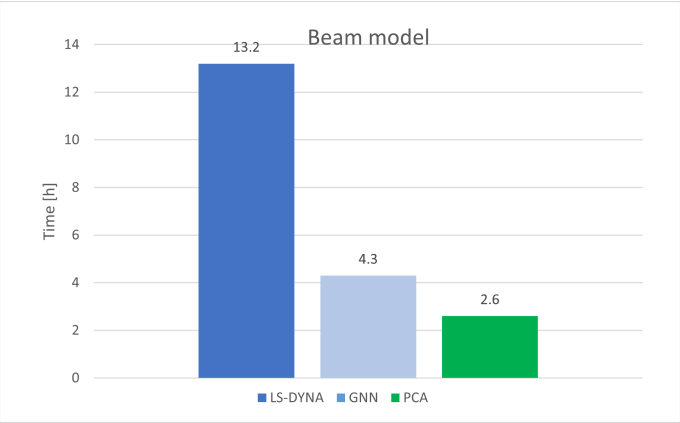


Figure 5.2: Time comparison to reproduce 500 simulations.

Specifically, 13.19 hours would be required from LS-DYNA. The time required by the solver to produce 100 simulations would be of 2.3 hours. The GNN requires 100 minutes only to train the autoencoder. The time for computing PCA is 1.7 seconds. As a result, the fastest would be PCA with 2.6 hours, namely the 19% of the time required by the LS-DYNA solver.

Pre-crash Simulation

For the pre-crash simulation, the LS-DYNA solver with 4 CPU cores and 16 GB of RAM required 65 minutes to run a single simulation.
The number of epochs chosen for the GNN was 500. Each epoch required 45 seconds with Google Colab Pro Tesla T4 GPU. CNN required only 20 seconds per epoch, for a total of 2.7 hours. PCA, in contrast, required only 6 minutes and 46 seconds, using the CPU with 16 GB available. In Figure 5.3, a comparison between GNN, CNN, and PCA training times are shown.

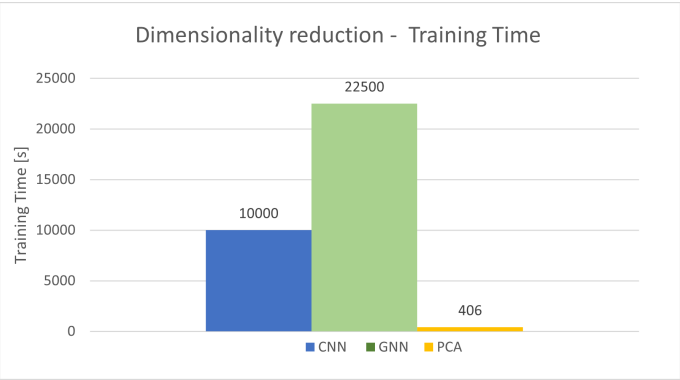


Figure 5.3: Comparison of training time between GNN, CNN, and PCA.

As for the beam model, the trend is the same. The GNN reveals to be the more time-consuming, with a training time of 6 hours. Training CNN, instead, takes less than half of the time.

As a base ground, let´s use the time calculated for the same set-up used to reproduce the results in the section above. The regression model was trained on 60 samples dataset. The test samples predicted were 15.
To assess if the time was saved with dimensionality reduction, let´s assume to produce 500 simulations. LS-DYNA solver would require 108 hours to produce 100 simulation to train the Dimensionality Reduction methods. To train GNNs and PCA, it would take 625 and 11 minutes respectively.

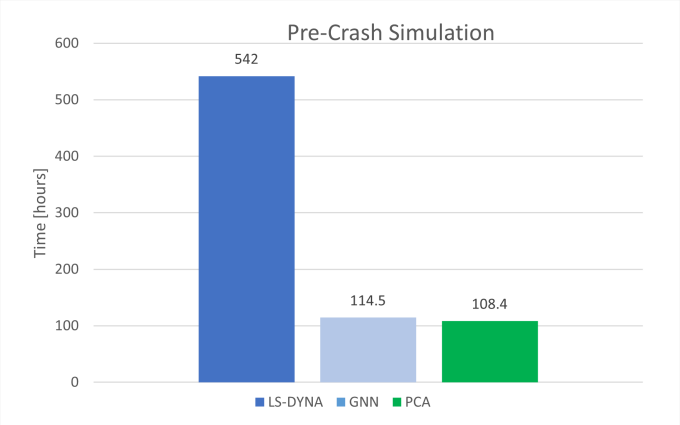


Figure 5.4: Pre-crash simulation: Comparison of training time between methods used and LS-DYNA

To reproduce 500 simulation with LS-DYNA solver would require 542 hours (about 22 days). The training time for the GNN would increase up to 10 hours. Whereas, interestingly, the PCA would be slightly under GNN with 108 hours.

6
Conclusion

This thesis aimed to investigate and research the potential of Graph Neural Network in predicting the nodal kinematics history of a pre-crash simulation. A machine learning pipeline has been developed to extract nodal history and the model´s geometrical information. It was trained and paired together with a supervised regression to create a surrogate of reduced dimension.

The reduced dimension was first obtained by creating an embedded space. Different methods such as PCA and CNN were compared in accuracy and performance. Moreover, different supervised tasks were studied to achieve better performance.

Particularly:

- Graph Neural Network showed the ability to predict pre-crash kinematics when paired with ML method. The best combination was given when using Gradient Boosting regression. It also showed to be a costly algorithm in terms of time and resources compare to PCA.

- When compared to an equivalent CNN, GNNs proved to achieve similar results. As a result, including the FE mesh information in the GNN did not improve significantly the outcome. Moreover, CNN are faster and less computationally demanding.

- On the other hand, for this application PCA proved to achieve a better result when compared to both CNN and GNN in both time and lower test error. When used together with Random Forest the reconstructed animation has proven to be the closest to the ground truth.

7

Future Work

The Hybrid III dummy model used was composed of 7443 nodes. In a full-HBM model the number of nodes can increase up to a million. The implementation of methods such as CNN and PCA can be cumbersome and limited.
The attention and the focus on Graph Neural Networks have established only in recent years. It is a very novel field, and in the future, new efficient and high-performing methods will be implemented. The potential of Graph networks is high as they are designed to work with unstructured and big datasets. As a result, the possibility to resume this method for similar applications is not yet concluded. However, as it is right now, there are various limitations, as mentioned in the Discussion such as graph pooling.

If instead, we look at the project architecture, a few things could be implemented in the future that could influence the outcomes. First of all, not every element or part of the Dummy model have been saved in the adjacency list. For example, the constrained parts or joints, even though are part of the model and have a fixed distance to other nodes, were not included in the adjacency list. Therefore they resulted being isolated nodes and did not participate in convolution. Moreover, other elements of the model could be included such as the seatbelt or the interior of the vehicle. Furthermore, the features of the nodes could be extended. Instead of saving only the x,y,z displacement, the velocity or the material properties of the node could be included and maybe improve the result.

Lastly, one more issue can be further investigated. During the animation in META, a distortion was observed when animating GNN reconstruction. It appeared as noise in the space domain, deforming the mesh pattern in a zig-zag shape, as shown in Figure 7.1 and 7.2. In LS-DYNA environment, there is similar phenomenon called hourglass. It results when producing a simulation and it is related the usage of under integrated elements. However, in this case, the hourglass-like behaviour was not witnessed in the original simulations, and therefore cannot be related to element integration, as it was observed only when reconstructed from the surrogate model. The hourglassing-resemblance phenomenon was observed in two different positions on the dummy, whereas the beam was entirely affected.

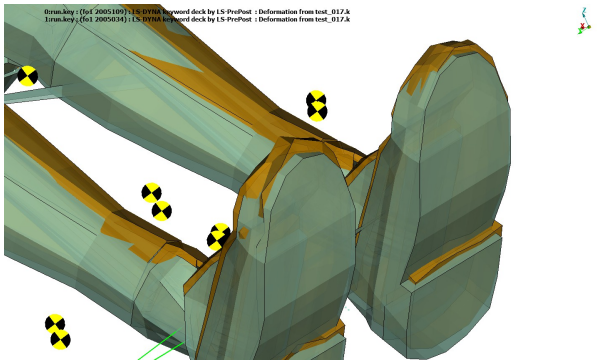


Figure 7.1: Hourglass-like anomaly observed on the right foot of the model

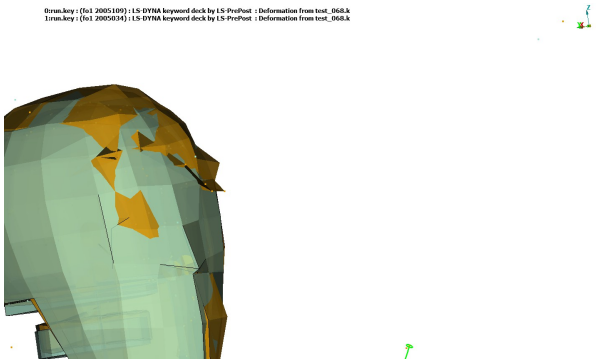


Figure 7.2: Hourglass-like anomaly located in the forehead of the dummy

Bibliography

[1] J. Howard and S. Gugger, "Deep learning for coders with fastai & pytorch: Ai application without a phd," 2020.

[2] A. for Safe International Road Travel, "Annual global road crash statistics."

[3] European Commission, "Road Safety Country Overview - Sweden," tech. rep., Directorate General for Transport, 2017.

[4] T. Xu, X. Sheng, T. Zhang, H. Liu, X. Liang, and A. Ding, "Development and validation of dummies and human models used in crash test," *Applied Bionics and Biomechanics*, 2018.

[5] WHO, "Global Status Report on Road," *World Heal. Organ.*, 2018.

[6] A. Sobhani, W. Young, and D. Logan, "Exploring the relationship of conflict characteristics and consequent crash severity," *ATRF 2011 - 34th Australasian Transport Research Forum*, 2014.

[7] W. Haddon, "The precrash, crash, and postcrash parts of the highway safety program," *SAE Technical Papers*, 1968.

[8] S. Gurram, V. Sai, and K. Reddy, "Vehicle occupant kinematics prediction using machine learning," 2021.

[9] IBM, "What is machine learning?." https://www.ibm.com/cloud/learn/machine-learning, Accessed: 30.09.2021.

[10] accessed on 2021-10-01.

[11] A. Agnihotri and N. Batra, "Exploring bayesian optimization," *Distill*, vol. 5, 05 2020.

[12] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[13] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, *Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs*. July 2017.

[14] A. Geron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 2019.

[15] J. Howard and S. Gugger, *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 2020.

[16] L. Hardesty, "Explained: Neural networks," 2017. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, Accessed: 22-03-2022.

[17] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015.

[18] J. Delua, "Supervised vs. unsupervised learning: What's the difference?," 2021. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning, accessed: 21 July 2022.

[19] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[20] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, 2021.

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[23] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *CoRR*, vol. abs/1509.09292, 2015.

[24] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[25] H. Mac, D. Tran, and H.-A. Tran, "Detecting attacks on web applications using autoencoder," 12 2018.

[26] H. Tran, "A survey of machine learning and data mining techniques used in multimedia system," 2019.

[27] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] M.-H. Chiu, Y.-R. Yu, H. Liaw, and L. Hao, "The use of facial microexpression and tree-forest model for predicting conceptual-conflict based conceptual change," 2016.

[29] C. Bakshi, "Random forest regression," 2020. https://www.capitalone.com/tech/machine-learning/how-to-control-your-xgboost-model/, accessed: 25 July 2022.

[30] I. Baturynska and K. Martinsen, "Prediction of geometry deviations in additive manufactured parts: comparison of linear regression with machine learning algorithms," *Journal of Intelligent Manufacturing*, vol. 32, 01 2021.

[31] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016.

[32] S. Pardy, "How to control your xgboost model," 2020. https://www.capitalone.com/tech/machine-learning/how-to-control-your-xgboost-model/, accessed: 26-07-2022.

[33] K. Kontolati, D. Loukrezis, D. G. Giovanis, L. Vandanapu, and M. D. Shields, "A survey of unsupervised learning methods for high-dimensional uncertainty quantification in black-box-type problems," *Journal of Computational Physics*, vol. 464, p. 111313, 2022.

[34] "Lasso python library, *Available* at https://www.lasso.de/en." Accessed on 04-08-2022.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Pas-

sos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd International Conference on Knowledge Discovery and Data Mining*, 2019.

[37] Livermore, "Ls-dyna® keyword user's manual." Accessed on 06-08-2022.

[38] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *CoRR*, vol. abs/2110.05292, 2021.

[39] J. Östh, K. Brolin, and D. Bråse, "A human body model with active muscles for simulation of pretensioned restraints in autonomous braking interventions," *Traffic injury prevention*, vol. 16, no. 3, pp. 304–313, 2015.

[40] J. Östh, K. Brolin, S. Carlsson, J. Wismans, and J. Davidsson, "The occupant response to autonomous braking: a modeling approach that accounts for active musculature," *Traffic injury prevention*, vol. 13, no. 3, pp. 265–277, 2012.

[41] K.-J. Larsson, B. Pipkorn, J. Iraeus, J. Forman, and J. Hu, "Evaluation of a diverse population of morphed human body models for prediction of vehicle occupant crash kinematics," *Computer Methods in Biomechanics and Biomedical Engineering*, pp. 1–31, 2021.

[42] S. M. Dunn, A. Constantinides, and P. V.Moghe, "Chapter 3 - concepts of numerical analysis," in *Numerical Methods in Biomedical Engineering*, Biomedical Engineering, pp. 65–83, Burlington: Academic Press, 2006.

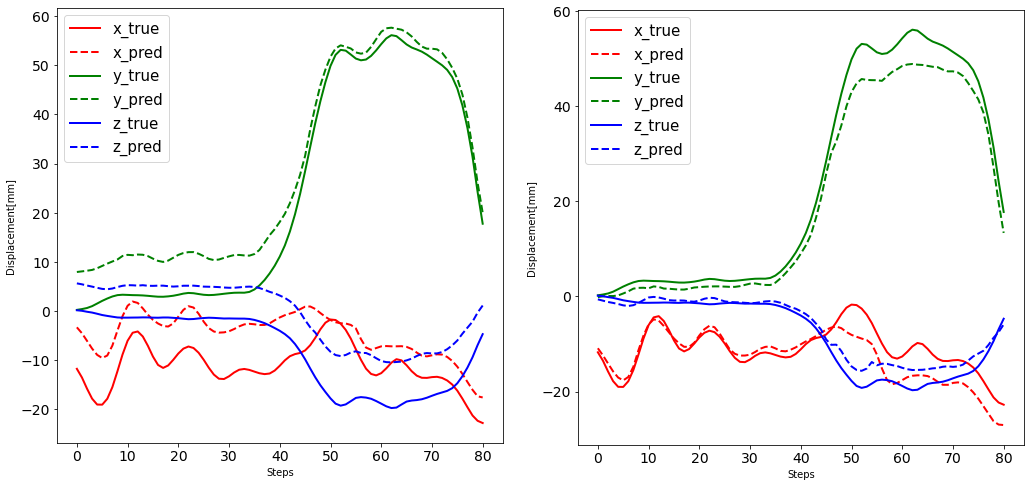
[43] M. Nguyen, D. Elder, J. Bayandor, R. Thomson, and M. Scott, "A review of explicit finite element software for composite impact analysis," *Journal of Composite Materials*, vol. 39, pp. 375–386, 02 2005.

[44] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," *CoRR*, vol. abs/2007.02133, 2020.

A

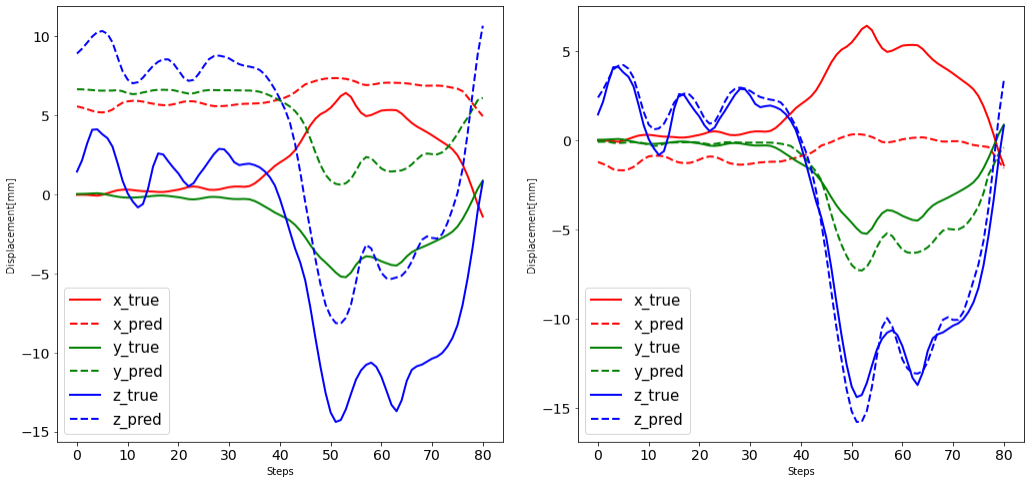
Appendix

Max-time scaling vs. Standardization In the figures below, the first column represented nodes plotted by scaling the dataset by the maximum timestep increase, while the second column displayed the same node with z_score standardization. The nodes displayed belong to different part of the Hybrid III Dummy, in particular a sample of the head, chest, left hand, right foot nodes were selected.



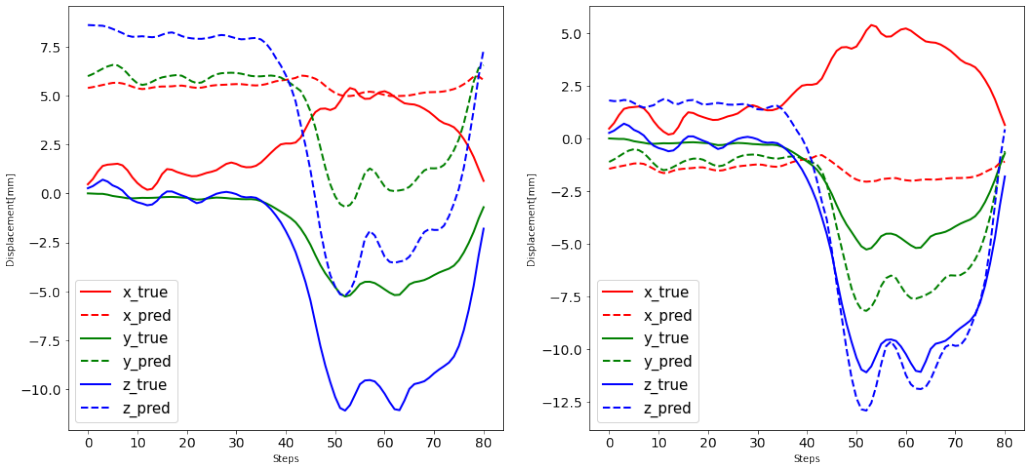
(a) Max timestep scaling **(b)** Standardization

Figure A.1: Head node - 2005184



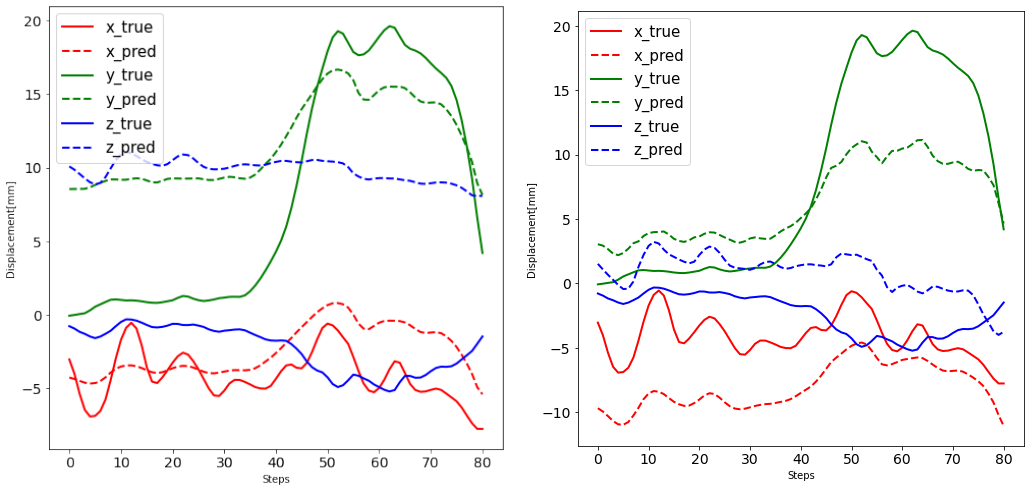
(a) Max timestep scaling **(b)** Standardization

Figure A.2: Chest Node - 2002195



(a) Max timestep scaling **(b)** Standardization

Figure A.3: Left hand 2002263

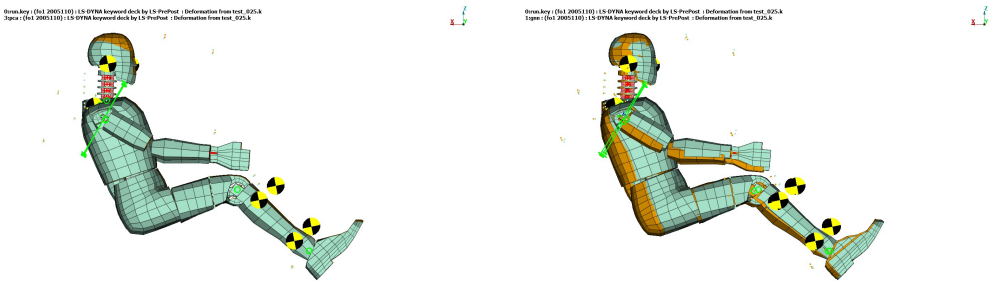


(a) Max timestep scaling **(b)** Standardization

Figure A.4: Right foot - 2009159

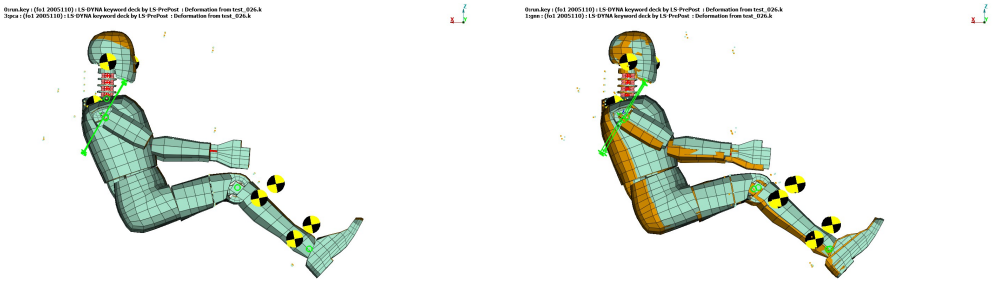
Animated Simulations In this section, frames from the 25th to the 29th from the animated simulation are shown. In contrast to the frame shown in the Result, here, a sequence of subsequent frames is displayed. The first comparison is between PCA and GNN, while the second compares GNN and CNN.

PCA - GNN As mentioned in the Results chapter in Section 4.2.2 the main difference in the animation is located in both upper and lower limbs, in the head, and in the pelvis area.



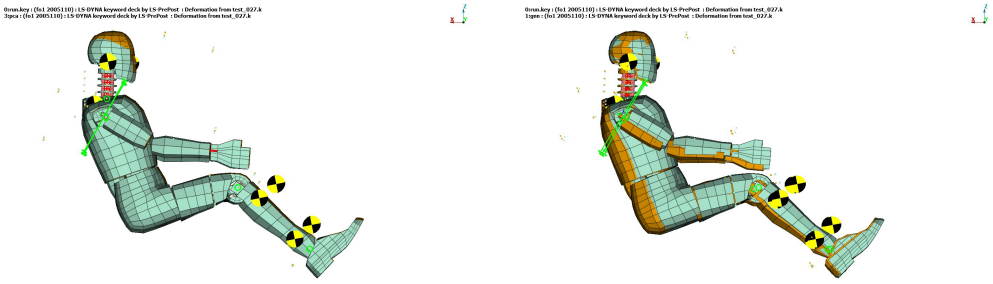
(a) PCA-GB **(b)** GNN-RF

Figure A.5: PCA-GNN comparison: after 929 ms



(a) PCA-GB **(b)** GNN-RF

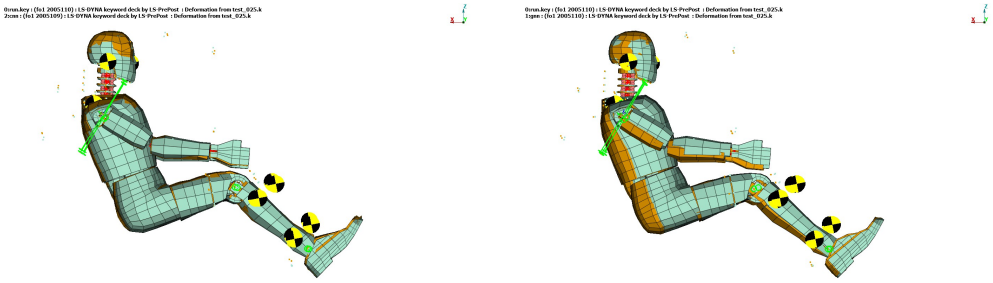
Figure A.6: PCA-GNN comparison: after 954 ms



(a) PCA-GB **(b)** GNN-RF

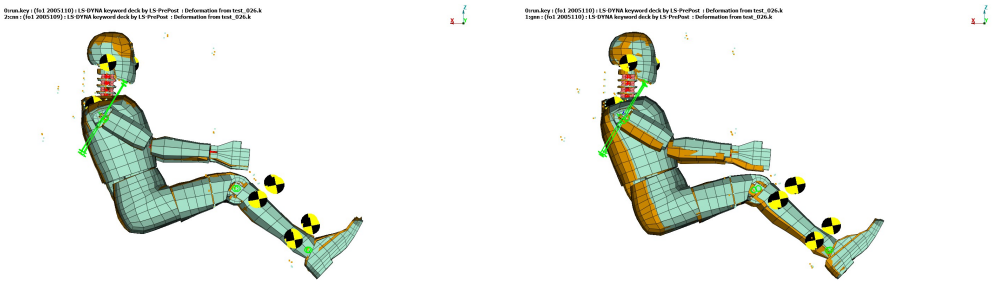
Figure A.7: PCA-GNN comparison: after 979 ms

CNN - GNN As for the previous set of animations, the same holds true. The main error is located in the extremities, where the maximum motion range is located. On the left the frames for the CNN, whereas on the right, the frames from GNN are displayed.



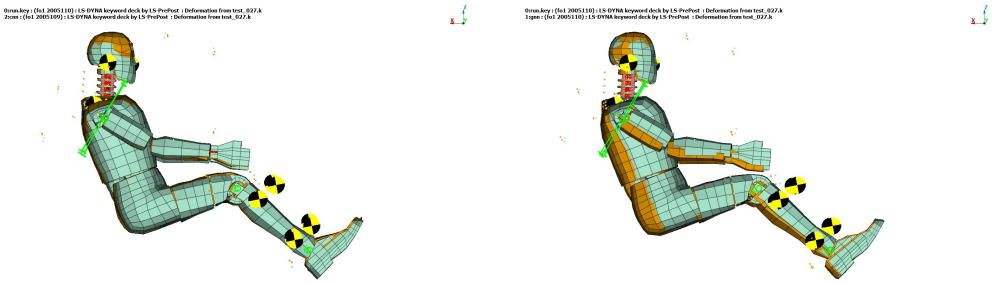
(a) CNN-GB **(b)** GNN-RF

Figure A.8: CNN - GNN comparison: after 929 ms



(a) CNN-GB **(b)** GNN-RF

Figure A.9: CNN - GNN comparison: after 954 ms



(a) CNN-GB **(b)** GNN-RF

Figure A.10: CNN - GNN comparison: after 979 ms

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY