DLR-IB-SY-SD-2023-142

Data-based leakage detection and uncertainty quantification in the manufacturing of large-scale CFRP components

Studienarbeit

Sreerag V Naveenachandran

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Institut für Systemleichtbau

DLR-IB-SY-SD-2023-142

Data-based leakage detection and uncertainty quantification in the manufacturing of large-scale CFRP components

Zugänglichkeit:

Stufe 2 – DLR-intern zugänglich: Der Interne Bericht wird elektronisch in ELIB abgelegt, erhält aber die Zugriffsbeschränkung „nur DLR-intern zugänglich“. Die bibliographische Beschreibung ist, wie bei allen ELIB-Einträgen, weltweit sichtbar.

Stade, *Oktober, 2023*

Der Bericht umfasst: 53 Seiten

Abteilungsleiter:
Dr.-Ing. Daniel Stefaniak

Autor:
Sreerag V Naveenachandran

Autor 2 / Betreuer:
Dr. Christoph Brauer

Deutsches Zentrum für Luft- und Raumfahrt

Institut für Analysis und Algebra

Studienarbeit

Data-based leakage detection and uncertainty quantification in the manufacturing of large-scale CFRP components

Sreerag V Naveenachandran

October 16, 2023

Institut für Analysis und Algebra
Prof. Dr. Timo de Wolff

Supervisor:
Dr. Christoph Brauer

Acknowledgements

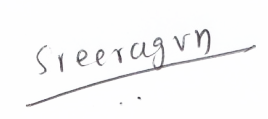
I would like to thank my supervisor, Dr. Christoph Brauer, for guiding me throughout this project and for his insightful lectures. I want to convey my gratitude to my examiner, Prof. Dr. Timo de Wolff, for providing me with profound insights during the courses machine learning with neural networks (MLNN) and advanced data science lab (FPDS). I would like to extend my heartfelt appreciation to my colleagues at the German Aerospace Centre (DLR) who provided invaluable support during the data acquisition process. My special thanks go to all the professors at the Institute for Analysis and Algebra, TU Braunschweig, for providing very knowledgeable courses in machine learning and data science.

Declaration

I certify that I have written this report without outside help and without using sources other than those indicated, and that the report has not been submitted in the same or similar form to any other examination authority and has been accepted by them as part of an examination performance. All statements that have been taken over verbatim or in spirit are marked as such.

Braunschweig

October 16, 2023



Sreerag V Naveenachandran

Abstract

Data-based leakage detection and uncertainty quantification in the manufacturing of large-scale CFRP components

Autoclave processing stands as a pivotal technique for manufacturing aircraft components from carbon fiber reinforced polymer parts (CFRP). This process involves curing stacked and impregnated materials with the application of heat and pressure within an airtight vacuum bag. The maintenance of vacuum integrity holds paramount importance in ensuring the top-notch quality of the components being produced. Leakages can result in porosity and voids making the final product unfit for use. While detecting the presence of leakage in the vacuum bag through residual mass flow is relatively straightforward, accurately determining the location of the leakage poses a significant challenge. Various methods [1, 2, 3] employed for leakage detection in industrial processes heavily relies on time-consuming manual labor and expert craftsmanship, resulting in considerable production delays. The objective of this project is to develop a machine learning-based methodology that effectively leverages flow rates measured at vacuum ports to precisely and efficiently localize leakages in industrial scale vacuum setups. Initially, we employ a deep neural network to address the scenario of a single leak in the vacuum bag. Subsequently, we investigate various methods like multi-task learning, masking etc. to handle cases involving multiple leaks in vacuum bags. Extensive experiments demonstrate promising results for our multi-leak model in addressing scenarios with multiple leaks.

Keywords : Artificial neural network, uncertainty quantification, leakage detection, multi-task learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autoclave processing is a crucial step in the manufacturing of aircraft components using carbon fiber-reinforced polymer parts (CFRP). This intricate process involves curing stacked and impregnated materials through the application of heat and pressure within an airtight vacuum bag [3]. The schematic setup of a vacuum bag, depicted in Figure 1.1, includes various auxiliary materials. These bags play a key role in applying uniform pressure to the component throughout the curing cycle, creating a closed system that isolates the component from the autoclave atmosphere.

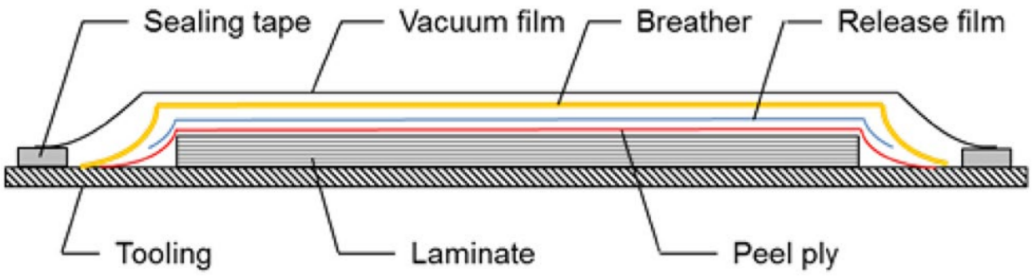


Figure 1.1: Schematic setup of a vacuum bag [3]

However, the success of curing processes can be hindered by potential vacuum bag leakages. If a leakage occurs during the curing cycle, critical defects in the composite material, such as porosity and voids, may result, rendering the final product unusable unless the issue is promptly resolved [1]. Therefore, ensuring vacuum integrity is crucial for the production of high-quality components [2].

While detecting leakage in the vacuum bag is straightforward through residual mass flow measurements with vacuum pumps, determining the precise location of the leakage poses a significant challenge. Various methods, as illustrated in Figure 1.2, are employed for leakage detection in aircraft industry. However, these traditional approaches are time-intensive and labor-dependent, leading to additional costs and significant production delays resulting in leakage-associated costs of approximately € 50.5–70.5k per year [3].

Figure 1.2: Leakage detection in vacuum bags: After the evacuation process, a pressure gauge is used to check for leaks by disconnecting the component from the vacuum and monitoring pressure changes in the vacuum bag as shown in right image. In older aircraft programs, a sensitive microphone is used for this purpose as in left image. Newer programs employ helium leak testing and dye tests to pinpoint leaks accurately [3].

Four computational methods for leakage localization were explored [1], namely

- Potential flow
- Numerical regression
- Quadrilateration
- Volume flow rate (VFR) matching
- Machine learning

Potential flow, while effective, revealed limitations near the edge. Conversely, machine learning approaches demonstrated notable potential for

robust and versatile leakage localization, particularly in managing multiple leakages [1]. The observation that the machine learning approach could be extended to address multiple leakages holds promise for detecting and localizing leaks in industrial-scale vacuum setups. This has the potential to decrease delays and ensure the production of aircraft components that are free from defects [1].

Additionally, the paper [4] explores incorporating prior knowledge of equivariance into the machine learning pipeline. Although implementing the equivariant architecture in our study on a flight wing structure is infeasible due to its asymmetric nature, informed machine learning combined with VFR data remains a promising avenue for improving accuracy and efficiency in leak localization during autoclave processing.

Deep learning, a powerful subset of machine learning methodologies, has garnered widespread acclaim in supervised learning for its robust and adaptable framework. The effectiveness of deep learning models lies in their capacity to represent intricate functions by integrating additional layers and units within a network. A prime illustration is convolutional networks, a specialized variant of feedforward networks that excels in object recognition from images. These networks employ hierarchical layers to autonomously learn and extract features at varying levels of abstraction, facilitating proficient image analysis and pattern recognition. Evolving from feedforward networks, recurrent networks assume a crucial role in diverse natural language applications [5].

Chapter 2 of this report delves into the theoretical foundations of our deep learning approach for modeling. It explores the network framework, the training process of the feedforward neural network, the dynamics of loss functions, and the pivotal importance of selecting appropriate input features for optimal model training.

The objective of this project is to explore and propose a novel, machine learning-based methodology that utilizes flow rates measured at vacuum ports during the autoclave process to achieve localization of leakages in industrial-scale vacuum setups. By reducing dependency on labor expertise and embracing automation, this project aims to streamline the leak detection process, thereby improving efficiency, reducing costs, and ensuring the delivery of high-quality aircraft components.

The student specialization project sets out to achieve the following goals:

1. **Data acquisition**: Description of the experimental setup and the data acquisition process.
2. **Data preparation**: Exploration of the data, identification of the suitable preprocessing steps and possible augmentation of the data synthetically.
3. **Model training**: Design of a neural network architecture to predict a single pair of leakage coordinates and use hyperparameter tuning to identify a final configuration.
4. **Uncertainty quantification**: Implementation of at least one existing method to quantify [6, 7] and visualize the uncertainty of the model's predictions.
5. **Multi-leakage detection**: Adaptation of the neural network for multi-leakage case.

Chapter 2

Deep learning

Deep learning, a potent facet of machine learning methodologies, revolves around artificial neural networks renowned for their representation learning capabilities. These architectures offer flexible learning approaches, spanning supervised, semi-supervised, or unsupervised techniques. The application of deep learning extends across diverse domains, including computer vision, speech recognition, machine vision, medical image analysis, social network filtering, and gaming. The remarkable performance of deep neural networks (DNNs), recurrent neural networks, and convolutional neural networks (CNNs) in these domains often rivals or exceeds human expertise [5].

In *supervised* deep learning, algorithms learn from a training set comprising input-output pairs $(\mathbf{x}, \mathbf{y})$, establishing associations between inputs and their corresponding outputs to approximate the underlying mapping function. This understanding enables the model to make accurate predictions for output variables $(\mathbf{y})$ when presented with new input data $(\mathbf{x})$. In contrast, *unsupervised* learning involves algorithms that engage solely with input features without explicit supervisory signals. Such approaches find utility in tasks like density estimation, sample generation, data denoising, identifying data manifolds, or clustering related examples [5]. This project predominantly focuses on supervised learning, employing regression to map input variables to output variables.

This chapter delves into the fundamentals of deep *feedforward networks* and explores the intricacies of neural network training. Various loss func-

tions and preprocessing techniques are examined, and hyperparameter tuning for optimizing network efficiency is thoroughly discussed. The chapter also explores the multi-task learning architecture implemented to address the challenge of multiple leakages. Finally, various methods for integrating bayesian uncertainty into deep learning methodologies are explored.

2.1 Neural network

Deep feedforward networks, also referred to as feedforward neural networks or multilayer perceptrons (MLPs), serve as the foundational architecture for deep learning models. The primary objective of a feedforward network is to approximate a target function f^*. For instance, in classification tasks, an input $\mathbf{x}$ is mapped to a specific category $\mathbf{y}$ by

$$\mathbf{y} = f^*(\mathbf{x}) \tag{2.1}$$

The essence of a feedforward network lies in learning optimal parameters $\boldsymbol{\theta}$ to achieve the most accurate approximation of the function, as defined by the mapping [5].

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}) \tag{2.2}$$

The model is represented by a directed acyclic graph illustrating the composition of functions. Consider a chain of three functions, $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$, forming the composition

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))) \tag{2.3}$$

During neural network training, the goal is to align $f(\mathbf{x})$ closely with $f^*(\mathbf{x})$. In summary, deep feedforward networks play a fundamental role in capturing complex relationships within data by fine-tuning parameter values for precise function approximations [5].

To ensure a deep learning model's capability extends beyond linear function modeling, non-linearity must be introduced. Most neural networks achieve this by utilizing an affine transformation controlled by learned parameters,

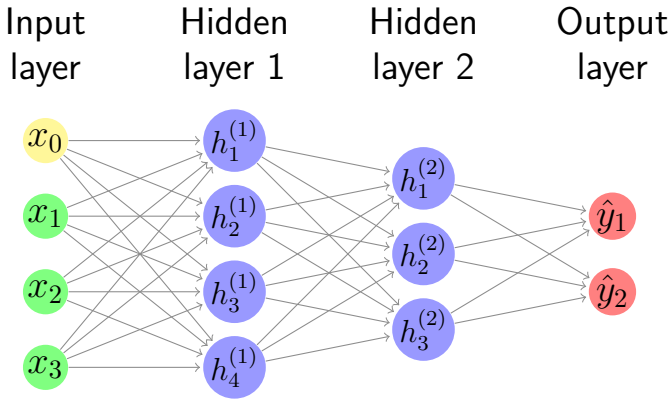


Figure 2.1: The Multilayer perceptron (MLP) illustrated in the figure comprises one input layer, two hidden layers, and one output layer. Each unit within the hidden layer is fully connected to the input layer, while the output layer is fully connected to the hidden layer [8]. Each layer could be considered as a non-linear mapping of vectors

followed by a fixed nonlinear function called an activation function [5]. In this context, an affine transformation is defined as

$$h = g(W^T \mathbf{x} + c) \tag{2.4}$$

where W represents the weights of linear transformation and c represents the biases. The rectified linear unit (ReLU) and its variations, known as ELU etc., have emerged as the default recommendation for the activation function $g()$ [9].

The *forward pass* in a neural network refers to the process of feeding input data through the network layers to generate an output. This involves multiplying the input data by the weights, adding biases, and applying activation functions at each layer [5]. The forward pass can be summarized as follows:

$$\begin{aligned} &\mathbf{a}^{[0]} := \mathbf{x} \\ &\left.\begin{aligned} &\mathbf{z}^{[\ell]} := \mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]} \\ &\mathbf{a}^{[\ell]} := \mathbf{g}^{[\ell]}(\mathbf{z}^{[\ell]}) \end{aligned}\right\} \text{ for } \ell = 1,\ldots\ldots,L \\ &\mathbf{y}_{\boldsymbol{\theta}}(\mathbf{x}) := \mathbf{a}^{[L]} \end{aligned} \tag{2.5}$$

where, $\mathbf{x}$ is the input and $\mathbf{y}_{\boldsymbol{\theta}}(\mathbf{x})$ is the output. This is illustrated in figure 2.1. In the case of regression, the output is typically unbounded, and therefore, no activation function is applied to the last layer. Alternatively, the activation can be assumed to be linear:

$$g^{[L]}(\mathbf{z}^{[L]}) = \mathbf{z}^{[L]} \tag{2.6}$$

2.1.1 Error backpropagation

The process of acquiring the weights and biases of a neural network is commonly referred to as *training*. This training is often framed as an optimization problem, as illustrated in equation (2.7), subject to certain constraints. The primary goal is to minimize a function known as the *loss* function, denoted as $J(\boldsymbol{\theta})$, within a parameter space Θ, while adhering to an equality constraint termed as a *regularizer*, denoted as $\Omega(\boldsymbol{\theta})$. The loss function gauges how well the neural network's predictions align with the actual target values. In regression tasks, this typically involves computing the *Mean Squared Error* (p = 2) or *Mean Absolute Error* (p = 1) between the network's output $\mathbf{y}_{\boldsymbol{\theta}}(\mathbf{x})$ and the actual ground truth $\mathbf{y}$. The regularizer in equation (2.7) acts as an artificial constraint to simplify the parameter space and is usually represented by the L^q norm of the network's weights [5].

$$\begin{aligned} \underset{\boldsymbol{\theta}\in\Theta}{\text{minimize}} \quad & J(\boldsymbol{\theta}) := \underset{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}{\mathbb{E}} \frac{1}{p} ||\mathbf{y}_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}||_p^p \\ \text{subject to} \quad & \Omega(\boldsymbol{\theta}) := \frac{1}{q} ||\mathbf{W}^{[\ell]}||_q^q \leq \tau, \quad \ell = 0, ..., L \end{aligned} \tag{2.7}$$

The loss function is computed based on the input/output pairs $(\mathbf{x}, \mathbf{y})$ from a distribution $\mathcal{D}$. Our goal is to minimize the expected loss arising from this distribution. In supervised learning, our objective is to identify a parameter set $\boldsymbol{\theta}^*$ that serves as a minimizer, fulfilling the following condition:

$$\boldsymbol{\theta}^* \in \underset{\boldsymbol{\theta}\in\Theta}{\arg\min}\, J(\boldsymbol{\theta}) \tag{2.8}$$

The optimization problem encountered in deep learning models is notably complex, being non-smooth, non-convex, and having a minimum of $\prod_{i=1}^{L-1} n_\ell!$ local minima, making it *NP-hard*. Gradient-based optimizers are used to manage this challenge. With an overwhelming count of optimization variables ($= \sum_{\ell=1}^{L} n_\ell n_{\ell-1} + n_\ell$), we often resort to first-order optimizers like *Stochastic gradient descent* (SGD) and its variants, since calculating second-order gradients can be prohibitive. These methods are designed to seek out local minima that, as empirical evidence suggests, are typically close to the global optimum in terms of function values [10, 11]. Hence, local optima generally provide satisfactory results for large networks. The gradient's analytical expression can be obtained with ease, but its numerical evaluation can be a computational burden. To address this, the back-propagation algorithm, or *backprop* [12], facilitates efficient gradient computation by propagating the error information backward through the network, significantly enhancing optimization speed [13] as shown in Algorithm 2.1.0.

Algorithm 2.1.0 Error backpropagation algorithm for a layered neural network represented as computation graph $G = (V, E)$.

(1) For a sample (x_n, y_n^*), propagate the input x_n through the network to compute the outputs $(v_{i_1}, \ldots, v_{i_{|V|}})$ (in topological order).

(2) Compute the loss $\hat{J}_n := \hat{J}(v_{i_{|V|}}, y_n^*)$ and its gradient

$$\frac{\partial \hat{J}_n}{\partial v_{i_{|V|}}}. \tag{2.9}$$

(3) For each $j = |V|, \ldots, 1$ compute

$$\frac{\partial \hat{J}_n}{\partial w_j} = \frac{\partial \hat{J}_n}{\partial v_{i_{|V|}}} \prod_{k=j+1}^{|V|} \frac{\partial v_{i_k}}{\partial v_{i_{k-1}}} \frac{\partial v_{i_j}}{\partial w_j}. \tag{2.10}$$

where w_j refers to the weights in node i_j.

The sensitivity of the optimization problem to initialization is high, with the convergence point being largely dependent on the initial starting point. To circumvent this, various techniques are employed to initialize the weights and biases of a neural network, often contingent on the chosen activation function within the network. A commonly utilized method is the He uniform variance scaling initializer [14], which assists in avoiding problems

related to exploding or vanishing gradients during optimization.

2.1.2 Loss function

The selection of the *loss function* is a pivotal aspect of any deep learning algorithm, influencing the model's performance and the interconnection of outputs within the network. This function gauges the model's effectiveness in predicting target values, with the objective of minimizing or maximizing it in machine learning. There is no universal loss function, as its choice hinges on various factors, including data characteristics, the presence of outliers, the selected machine learning algorithm, efficiency in gradient descent, derivative availability, and prediction confidence. Loss functions can be broadly categorized into two types: *classification* and *regression* loss [15].

Classification loss functions are applied to discrete class label prediction tasks, where they compare predicted class probabilities to the true class labels. Notable examples include *Binary Cross-Entropy* for binary classification and *Categorical Cross-Entropy* for multi-class problems. On the other hand, regression loss functions are employed in continuous numerical value prediction tasks, quantifying the difference between predicted and actual values. Common examples include *Mean Squared Error (MSE)* and *Mean Absolute Error (MAE)*.

2.2 Hyperparameter tuning

Machine learning models consist of two distinct types of parameters:

- Hyperparameters: These are parameters set arbitrarily by the user before the training process commences. Examples include the number of neurons in a neural network or the depth of the network. Hyperparameters significantly impact computational resources, such as the time and memory required for the algorithm's operation, as well as the efficacy and accuracy of the model in making predictions on new data.
- Model parameters: These parameters are learned during the model training process. They encompass weights in neural networks or

coefficients in linear regression. Model parameters define how the input data is utilized to generate the desired output and are adjusted through training.

Fine-tuning machine learning models involves optimizing hyperparameters, constituting an optimization problem where the objective is to discover the optimal combination of hyperparameter values to either minimize (e.g., loss) or maximize (e.g., accuracy) a specific function. Two prominent strategies for hyperparameter selection are manual and automatic. Manual tuning is effective when predefined starting points or substantial user experience in similar tasks are available. However, automated algorithms become necessary when such starting points are unavailable, aiming to discover appropriate hyperparameter values [5].

KerasTuner, a framework for efficient hyperparameter optimization [16], seamlessly integrates with TensorFlow [17]. It simplifies the complex process of hyperparameter search. A model builder function outlines the model architecture and defines the hyperparameter search space. This process involves tuning the number of layers, the number of units within each layer, and the learning rate for the optimizer. Ultimately, the model builder function returns a compiled model with the optimal set of hyperparameters.

The KerasTuner library offers various built-in search algorithms, including bayesian optimization, hyperband, and random search. Researchers can also extend it to experiment with new search algorithms. When dealing with three or fewer hyperparameters, a frequently employed method is *grid search*. The user selects a limited set of values for each hyperparameter, and the algorithm develops a model for every potential combination of hyperparameter values. An alternative, *random search*, provides a simpler implementation, user-friendliness, and expedited convergence to appropriate hyperparameter values. *Random search* eliminates the need for discretizing or binning hyperparameter values, allowing exploration of a larger set of values without additional computational costs.

The *model-based algorithms* for hyperparameter search often use bayesian regression models to predict the expected validation set error and its uncertainty for each hyperparameter. Optimization requires a balance between exploration, proposing hyperparameters with high uncertainty, and

exploitation, choosing hyperparameters that the model is confident about. This balancing act is crucial for efficiently discovering optimal hyperparameter values and enhancing model performance [5].

The *hyperband* algorithm, introduced in 2018, is designed to allocate computational resources efficiently throughout the hyperparameter search process. This algorithm segments hyperparameter configurations into brackets, assigning decreasing resources to each bracket. Configurations exhibiting poor performance are terminated early, while the top-performing ones advance to the next bracket with increased resources [18]. The *hyperband* method facilitates the swift identification of suitable hyperparameter configurations, leading to a reduction in overall computational expenditure.

2.3 Multitask learning

In our strategy, we aim to employ advanced techniques such as multi-task learning (MTL) to construct a model capable of addressing scenarios involving multiple leakages. Multi-task learning seeks to enhance generalization performance by leveraging domain-specific information from related tasks. This network architecture improves generalization by simultaneously training on multiple tasks while sharing a common representation, as depicted in Figure 2.2. The shared representation acts as an inductive bias, benefiting from the training signals of additional tasks to enhance overall performance. Essentially, MTL utilizes knowledge gained from one task to assist in learning other related tasks [19].

Various strategies are prominent in optimizing multi-task learning, including per-task loss weighting based on factors such as uncertainty or learning speed. Commonly employed regularization techniques involve L2 and trace norms. To prevent conflicting gradients between tasks, gradient modulation and replacement are utilized. Multi-objective optimization is also widely adopted. Methods for learning relationships between tasks include conducting large-scale empirical studies to identify tasks exhibiting positive learning dynamics when learned together, comparing network representations to assess task similarity, and learning task embeddings [19].

The architecture employed for the purpose of multiple leakage cases in

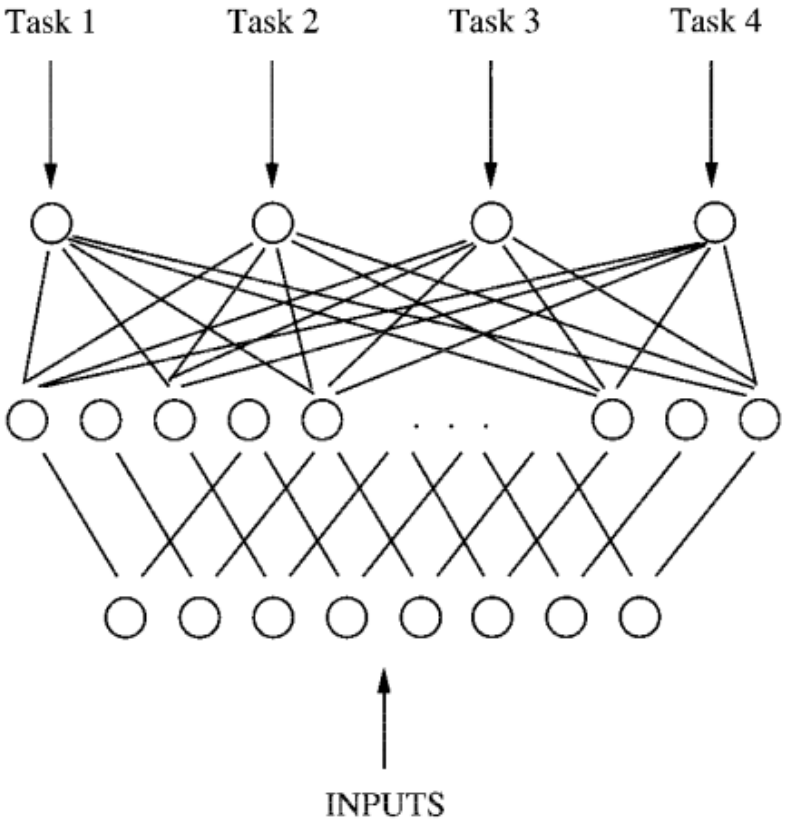


Figure 2.2: Multitask learning (MTL) of four tasks with the same inputs [19]

multi-task learning draws inspiration from the DarkNet/YOLO network architecture, as explained in Section 3.3.2 [20].

2.4 Uncertainty quantification

Deep learning has gained considerable popularity among researchers across various disciplines, including physics, biology, and manufacturing. Neural networks (NNs), convolutional neural networks (convnets), and other deep learning tools find extensive application in these domains. However, in fields where accurately representing model uncertainty is crucial—such as physics and biology—there exists a growing necessity to incorporate bayesian uncertainty into deep learning methodologies. This shift under-

scores the demand for more dependable and interpretable models capable of accommodating uncertainties in predictions. Standard deep learning tools designed for regression and classification tasks typically lack inherent mechanisms to capture model uncertainty [6].

The use of dropout and its variations in neural networks has been demonstrated to emulate a bayesian approximation of the gaussian process, a well-established probabilistic model [6]. Dropout acts as a conventional regularization technique during training to combat overfitting. This approach involves randomly deactivating (setting to zero) a portion of neurons in each training iteration, fostering the creation of more robust models.

To evaluate uncertainty in predictions, particularly within neural networks, dropout can be extended to remain active during inference or testing. Instead of generating a single prediction, this extended dropout process is repeated multiple times (often several hundred), each time with a distinct set of randomly deactivated neurons. Subsequently, the mean (average) and standard deviation of these prediction iterations are calculated to derive the final prediction. This standard deviation serves as an indicator of the inherent uncertainty in the model's predictions, providing valuable insights into the model's level of confidence or uncertainty regarding its predictions [6].

This strategy improves the dependability and interpretability of deep learning model predictions by furnishing valuable uncertainty estimates. By integrating dropout in both training and testing phases and leveraging monte carlo sampling, it becomes feasible to generate prediction intervals with minimal alterations to existing neural network architectures. This project investigates the application of dropout as a means to gauge the model's uncertainty.

Interval Neural Networks (INNs) present another approach for assessing predictive uncertainties in deep learning models [7]. INNs calculate not only point estimates but also uncertainty intervals for model predictions, delivering a metric for the model's confidence in its outputs. Through the incorporation of interval-based training and inference, INNs proficiently quantify model uncertainty.

2.5 Feature importance

Preprocessing and feature selection are crucial steps in the machine learning pipeline that can significantly impact the performance and effectiveness of a model. Feature importance is a powerful approach for selecting the most relevant features from a dataset. It not only enhances our understanding of the data but also aids in reducing model complexity and improving interpretability. Embracing feature importance methods, such as coefficient magnitude, decision trees and ensembles, permutation importance, LASSO regularization, and SHAP values, empowers us to gain valuable insights and make more informed decisions in the realm of machine learning [21]. This is explained in section 3.2.1.

2.6 Data normalization

Data normalization is another crucial data preprocessing step in machine learning, particularly when dealing with features that have different scales or units. This procedure greatly enhances the conditioning of the optimization terrain, predominantly the condition number of the Hessian. Decreasing the condition number of the Hessian facilitates more rapid convergence of the optimizer [5]. Generally, input data is normalized by recalibrating the mean μ to zero and the standard deviation σ to 1 as shown in Equation 2.11.

$$Standardization : X' = \frac{X - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{2.11}$$

2.7 Convex combination

A vector $\mathbf{v}$ in a real vector space is a convex combination of a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ if there exist scalars $a_1, a_2, \ldots, a_n$ such that:

$$\begin{aligned} &a_i \geq 0 \ \text{ for all } i, \\ &\sum_{i=1}^{n} a_i = 1, \\ &\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_n\mathbf{v}_n. \end{aligned} \tag{2.12}$$

If the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ belong to a convex set, then their convex combination $\mathbf{v}$ also belongs to that set [22].

Convex combination model serves as a benchmark (baseline model) for evaluating the performance of the model in the single leakage case. In this scenario, the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ represent the positions of the vacuum ports, and a_i denotes the measurements in each of the vacuum ports after appropriate transformations for optimal predictions. The method's advantage lies in its simplicity, speed, and lack of a learning requirement. However, a notable drawback of the convex combination model is its limitation in predicting leakages solely within the enclosed area formed by the vacuum ports.

Chapter 3

Experimental setup / Methodology

This chapter delves into the experimental setup established for data acquisition and the array of challenges encountered during this process. Additionally, it details the chosen data preprocessing steps and outlines the problem statements for both single-leak and multi-leak cases. The chapter also elucidates how the YOLO network inspired the approach to handle multiple leakages.

3.1 Data acquisition

In order to generate the required measurement data, an airtight setup was implemented on an Airbus A350 aircraft wing section with dimensions $[16048mm \times 5233mm]$. This configuration consisted of a breathable cloth and a bagging film. There were ten ports positioned on the wing, leading to vacuum pumps equipped with flow meters, while sealant tape was used to maintain an airtight seal around the structure's perimeter. Underneath the vacuum film, the presence of the breather material provided an air-conducting layer, preventing the formation of air pockets prior to full evacuation. Even though it was possible to measure the mass flow in each pump over time, our study primarily concentrated on examining the steady-state data - the residual mass flow that remained post full evacuation. An illustration of our experimental setup can be seen in Figure 3.1, as described in [4].

The initial step involves manually puncturing the airtight setup at a ran-

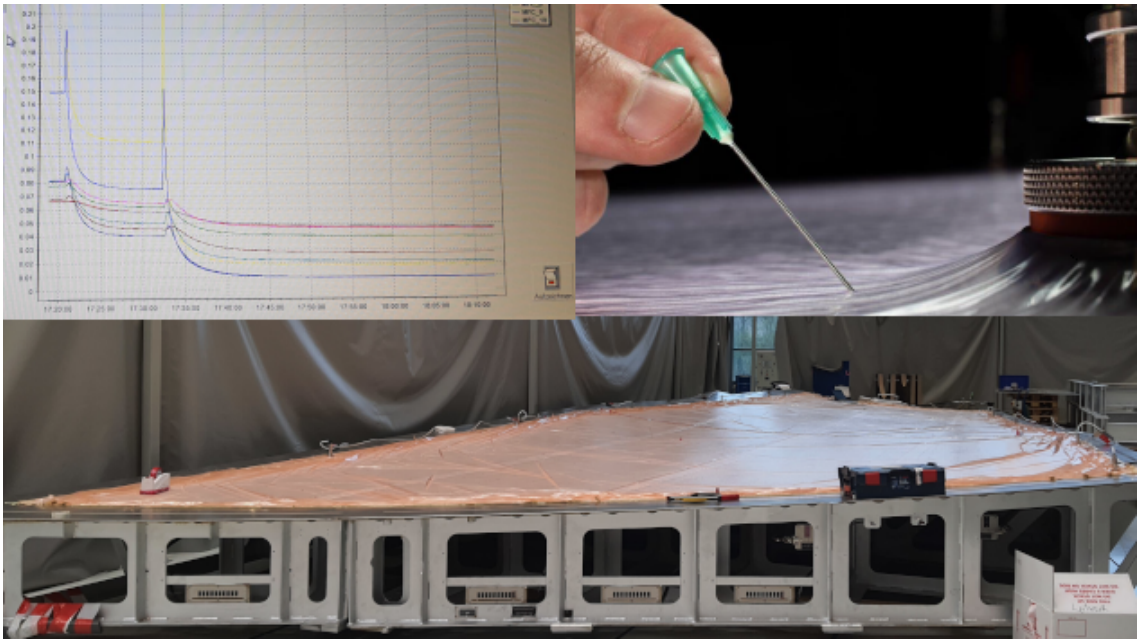


Figure 3.1: The airtight vacuum setup, shown at the bottom, is punctured by a hypodermic needle, as indicated at the top right. Volumetric flow meters, depicted at the top left, record the residual flows at the vacuum pumps during their steady state. These measurements serve as inputs $x \in \mathbb{R}^{10}$, which the model leverages to forecast the locations of the leakages.

domly selected point $y \in [0, 5233] \times [0, 16048]$ using a hypodermic needle to generate measurement data (see Figure 3.1 top right). Subsequently, the valves are sealed, and the remaining mass flow rates $\mathbf{x} \in \mathbb{R}^{10}$ at each of the ten vacuum pumps are recorded. This process is iterated, fixing each leak, until a total of 957 samples are compiled, encompassing 512 single leakage samples and 445 double leakage cases. However, the procedure encounters a challenge due to the constraint of limited data, given the time-consuming nature of this specific technique, amounting to approximately 250 hours in total.

The leakage coordinates which are target variables $\mathbf{y} \in \mathbb{R}^2$ of the training data and associated mass flow controllers (MFCs) $\mathbf{x} \in \mathbb{R}^{10}$ which are the inputs, are depicted in Figure 3.2.

During the data acquisition process, unintentional leakages occurred due to movement over the test piece, introducing noise into the data. Consequently, routine checks were essential to ensure the absence of undetected leakages and to maintain the integrity of the data. The data acquisition

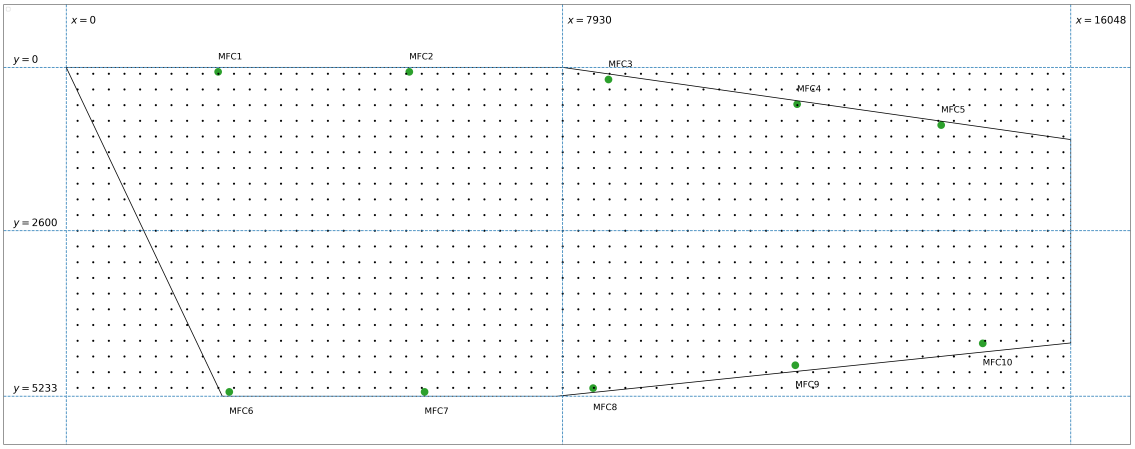


Figure 3.2: The image depicts the wing section of Airbus A350. The ten mass flow controllers are depicted in green. The grid serves to illustrate the planned locations for inducing leakages for data collection. The dimensions of the wing are annotated in the image, and within the rectangular box, the contour of the wing is outlined.

spanned a month, and exposure to sunlight resulted in unintended leakages on the vacuum, requiring rectification. Moreover, a malfunction in the vacuum pump during this period necessitated its substitution with a lower-power pump, given the unavailability of an equivalent-power pump. This replacement extended the waiting time required to reach a steady state and record the flow rate in each MFC, thereby causing further delays in the data acquisition process.

3.2 Data preparation

Scientific research heavily relies on the collection and analysis of relevant data. A crucial aspect of this process is feature selection, where a subset of input variables with the utmost relevance to the target variable is identified and chosen. Human involvement in data collection can inadvertently introduce errors into the data intended for training deep neural networks. Therefore, meticulous data sanitation and preprocessing are imperative.

Specifically, in a single leakage case, the pump proximate to the leakage should manifest the highest flow rate, as elucidated in Figure 3.3. By adhering to this principle, substandard data can be effectively filtered out manually for each sensor.

The cumulative flow rate across all pumps is closely monitored. Given the consistent size of the needle and the uniformity of the leakage during the data acquisition phase, only a minor variance in the total flow rate across all vacuum pumps is anticipated. This assumption is grounded in the stable conditions upheld throughout the data collection process.

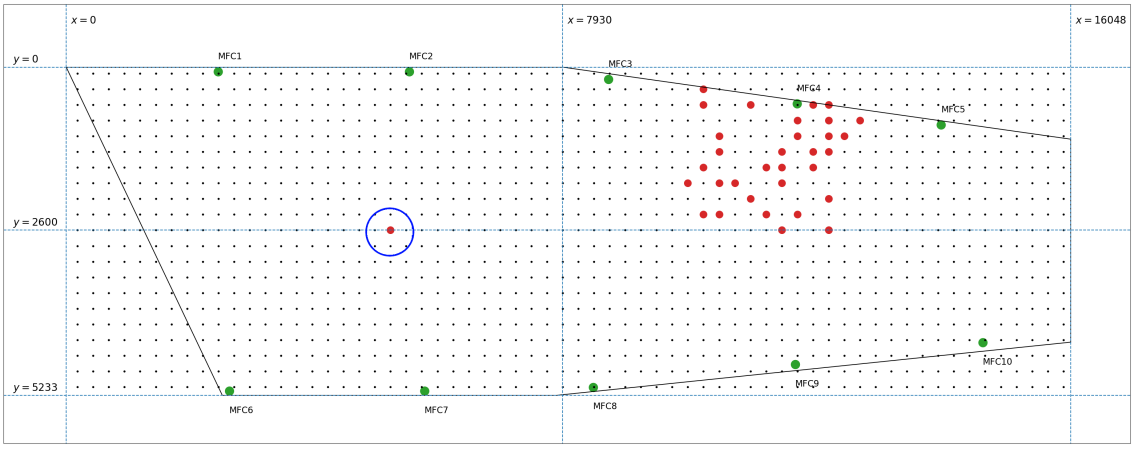


Figure 3.3: The image depicts the wing section of Airbus A350. Red dots represent the leakages, and green dots indicate the locations of mass flow controllers (MFC). In this particular image the leakages that exhibited the highest flow rates at sensor 4 are plotted. Therefore, one would expect the closest sensor to all these leakages to be sensor 4. The red dot with a blue circle around it signifies poor quality data, as it is not close to sensor 4, despite showing the highest leakage at this sensor.

Following the exclusion of low-quality samples, a total of 901 samples were retained, encompassing 475 single leakage instances and 426 double leakage cases. A seperate dataset is also prepared to hanlde the case of single-leak. The organized dataset were partitioned into three subsets — training, testing, and validation — adhering to a ratio of 75:15:15, respectively. The lack of a symmetrical structure poses a challenge to the feasibility of straightforward data augmentation techniques. To ensure consistency, all datasets underwent normalization using the mean and standard deviation of the training data, as detailed in section 2.6.

3.2.1 Feature selection

The observations indicated that the flow meters recorded flow values, even when there was no leakage (MFC1_residual to MFC10_residual) induced.

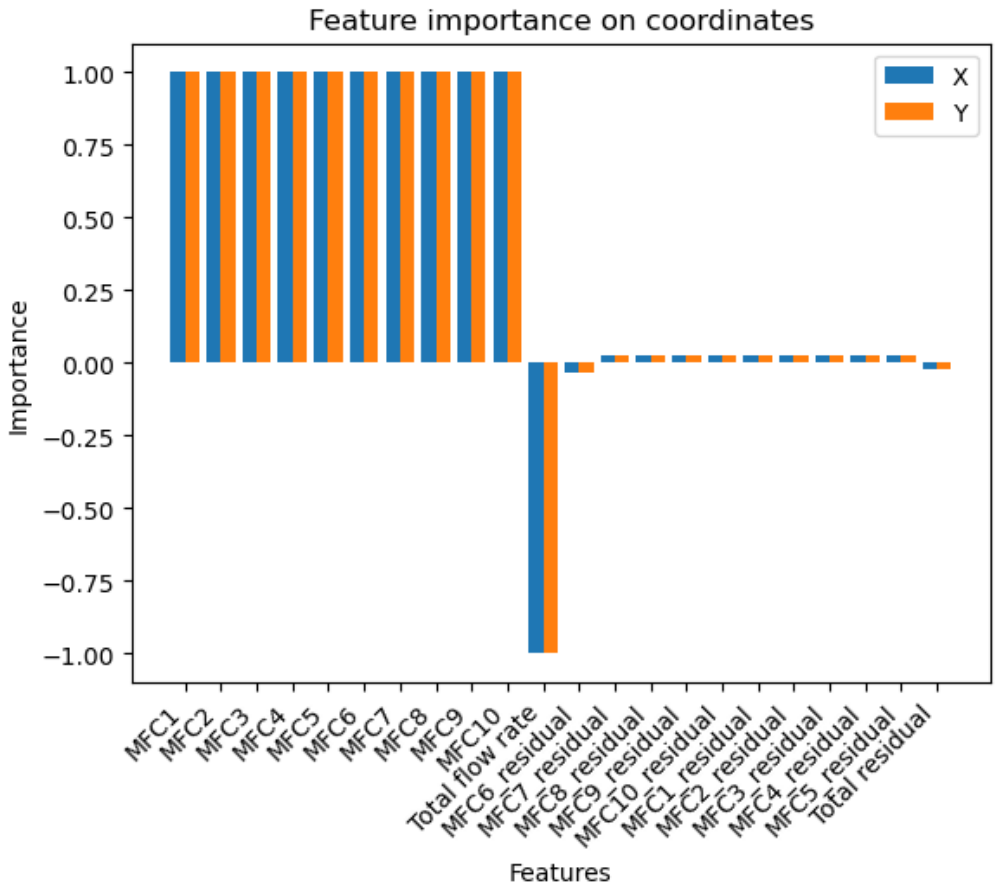


Figure 3.4: The figure illustrates the feature importance for the target variables y.

These readings largely remained unaltered throughout the entire experimental phase. Therefore, we deemed it necessary to evaluate if these features should be selected into the model training process. We took into account additional features such as total flow (Total flow rate) and total no-leakage-flow (Total residual). We then assessed feature importance utilizing techniques like XGBoost, random forest, and linear regression. Aside from the 10 mass flow rate parameters, all other features received minimal importance. In the Figure 3.4, the features MFC1 through MFC10 correspond to measurements obtained from the 10 mass flow controllers (MFCs) after manual leakage simulations. Remarkably, these MFC features consistently demonstrate the highest importance across various importance analysis algorithms. Additionally, the features MFC1_residual to MFC10_residual represent mass flow rates through the 10 MFCs in scenar-

ios without manual leakages. Finally, Total_flow_rate and Total_residual denote the total flow rates through the 10 MFCs when manual leakages are present and when no leakages are present, respectively. This outcome aligns with our expectations, as many of these features remained constant across multiple samples, leading us to either overlook them or assign them a low significance as shown in Figure 3.4.

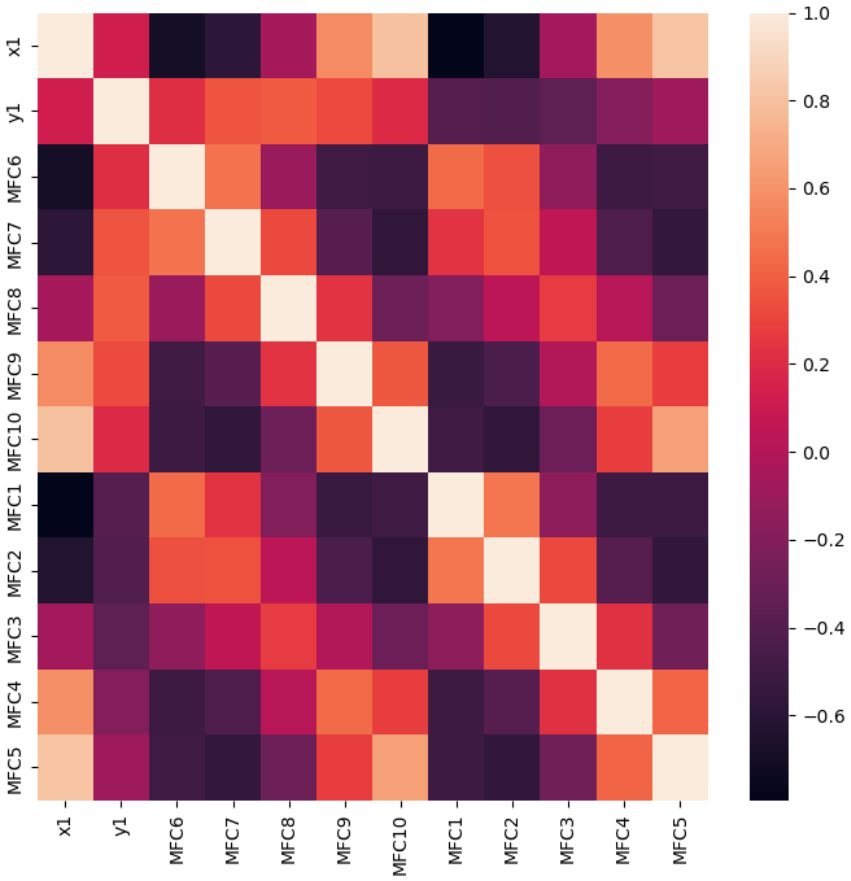


Figure 3.5: The heat map shows correlation of different possible input features among themselves and the correlation of the input features with the target.

In addition to the above, we trained a neural network to investigate whether hyperparameter tuning could yield a model that considers more features than just the 10 flow rate parameters as the most accurate. We aimed to examine whether expanding the feature set, combined with careful optimization of the model parameters, could potentially enhance model performance. However, our investigations found that the model with the

10 flow rate features consistently emerged as the top choice during the hyperparameter tuning process.

As demonstrated in Figure 3.5, most features exhibit a high correlation with the target variables. Features MFC8 and MFC3 show negligible correlation—neither positive nor negative—with the **x** coordinate of the target variable. Meanwhile, feature MFC5 demonstrates minimal correlation with the **y** coordinate of the target variable. Numerous other features are only moderately correlated with the **y** coordinate of the target variable, potentially indicating a non-linear relationship between these variables and the corresponding coordinate.

3.3 Network architecture

In an industrial context, it's possible to have multiple leakages in the vacuum bag, though the exact number of leakages may be unknown. We address this issue as two separate scenarios. Initially, we ascertain whether our model accurately predicts the instance of single leakage, as elaborated in section 3.3.1. Special attention is paid to the predictions along the edges of the wing to determine if our model surpasses the performance of existing computational models. Then, to depict the scenario of multiple leakages, we use data from both single and double leakage situations to train the multiple leakage model, as discussed in section 3.3.2.

3.3.1 Case I: Single leakage

We aim to learn a parameterized function $f_\theta : \mathbb{R}^K \to \mathbb{R}^N$ for which we have training data available

$$S := \left\{ \left(x^{(i)}, y^{(i)} \right) \mid i = 1, \ldots, m \right\} \subseteq \mathbb{R}^K \times \mathbb{R}^N \tag{3.1}$$

the input features **x** consists of ten normalized real values $(K = 10)$ which are the mass flow rates at ten pumps,

$$\mathbf{x} \in \Delta := \left\{ \mathbf{x} \in \mathbb{R}^{10} \mid x \geq 0 \right\} \tag{3.2}$$

and the target features **y**, consisting of two real values $(N = 2)$ which are two-dimensional coordinates of the position of leakage.

Out of 475 samples for single leakage case, 322 samples are used for training, 81 for validation and the remaining 72 for testing. The data set is split randomly.

3.3.2 Case II: Multi leakage

We aim to learn a parameterized function $f_\theta : \mathbb{R}^K \to \mathbb{R}^N$ for which we have training data available

$$S := \left\{ \left(x^{(i)}, y^{(i)} \right) \mid i = 1, \ldots, m \right\} \subseteq \mathbb{R}^K \times \mathbb{R}^N \tag{3.3}$$

, the input features x consists of ten normalized real values $(K = 10)$ which are the mass flow rates at ten pumps,

$$\mathbf{x} \in \Delta := \left\{ \mathbf{x} \in \mathbb{R}^{10} \mid x \geq 0 \right\} \tag{3.4}$$

and the target features y, consisting of four real values $(N = 4)$ which are two-dimensional coordinates of the position of leakage.

Out of 901 total samples, 630 samples are used for training, 145 for validation and the remaining 126 samples for testing. The data set is split randomly. Our dataset encompasses instances with both single and double leakages. The base model selected is a simple linear regression model with 4 outputs: the $(\mathbf{x}, \mathbf{y})$ coordinates of the two leaks.

The primary objective is to determine if, given a specific input, the model can accurately predict the coordinates of a single leakage when only one leak is present or predict the coordinates of both leakages when two are present. For this purpose we explored two different strategies.

- Preprocessing of the outputs by sorting
- A Multi task learning network inspired from the you only look once (YOLO) architecture.
- Masking of the loss for the second leakage for the samples where there is only one leakage.

3.3.2.1 Experiment 1 - Sorting of outputs

To address the challenge of handling multiple leakage cases, we employed various techniques and model architectures. Given that the order of the outputs is not significant, i.e., the sequence in which the coordinates of each leakage are presented to the output neuron, we augmented the training data by randomly swapping the order of the target outputs during training. The intention here is to aid the network in comprehending the equivalence of both outputs.

Another method explored involved arranging the output neuron values in a manner that places the leakage coordinates with the least x-coordinates in the first two neurons.

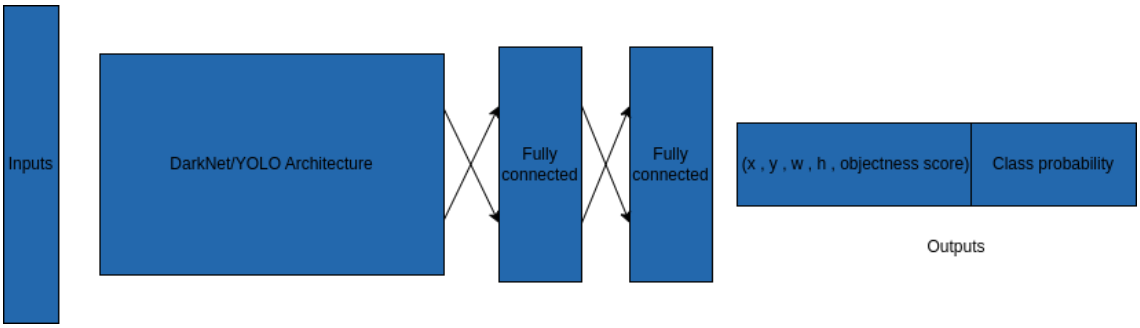


Figure 3.6: The architecture of the YOLO network, with its output layers featuring the "objectness" score. The ouput vector of the YOLO network includes coordinates (x, y) which represents the coordinates of the centre of the bounding box, w and h which represent the width and height of the bounding box, "objectness score" which represents the confidence that a predicted bounding box encloses an object regardless of its class and class probability.

3.3.2.2 Experiment 2 - Multi-task learning

The challenge of multiple leakages is redefined as a multi-task learning regression problem. In this setup, the inputs are directly transformed into the coordinates of each leakage, and class probabilities are used to identify the presence of an additional leakage in comparison to the single leakage scenario. Consequently, the model is designed with 5 output neurons. This approach is grounded in the idea that simultaneously learning the

number of leakages could potentially aid in refining the prediction of leakage coordinates. Borrowing from the You Only Look Once (YOLO) [20] network architecture (a popular multi-task learning object detection model known for its speed and accuracy) as shown in Figure 3.6, this structure confirms the capability to predict the number and location of leakages, if any exist. The output encompasses the coordinates (x, y) of the leak, together with confidence scores, referred to as "leakageness" score, much like the "objectness" score in YOLO [20], which represents the confidence that a predicted bounding box encloses an object regardless of its class. In this project, inspired from the objectness score of YOLO network, "leakageness" score is introduced which denotes the likelihood that a sample corresponds to a scenario with more than a leakage. Hence, for the sample with single leakage, the "leakageness score" is set to zero and for the case with 2 leakages, "leakageness score" is set to 1. A YOLO-inspired architecture designed for leakage detection is depicted in the Figure 3.7.

Two types of loss are considered:

1. **Localization loss:** This leakage location loss gauges the precision of the predicted leakage location coordinates against the actual leakage location coordinates, employing mean squared error (MSE).
2. **Confidence loss:** The network predicts a "leakageness" score for each sample, indicating the probability of the second leak. The confidence loss quantifies the error in predicting this "leakageness" score relative to the actual value, using binary cross-entropy loss.

For simplicity, we consider samples with up to two leakages. If the "leakageness" score is zero during training, indicating no second leak, the corresponding leakage coordinates (x, y) are usually set to fixed values predefined based on the dataset. This offers preliminary information to the model, even for cells without a leak. Therefore, in this case of no secondary leakage, the coordinates are set to the airplane wing's center.

3.3.2.3 Experiment 3 - Masking of loss

For scenarios where the second leakage is absent, two distinct approaches were explored. In one method, the coordinates were set to the airplane wing's center, and the model was trained accordingly.

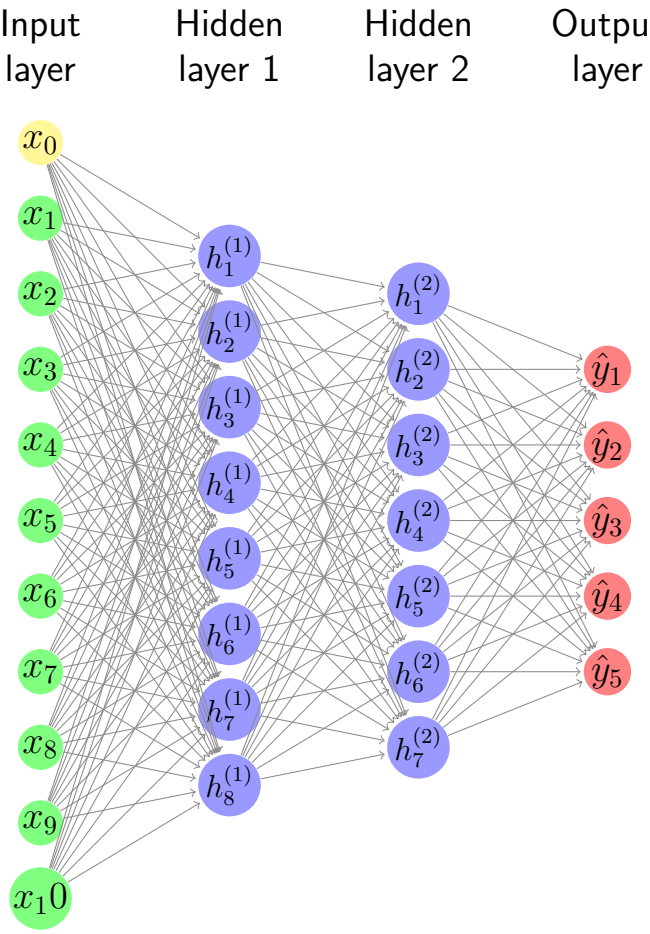


Figure 3.7: This multi-task learning architecture draws inspiration from You Only Look Once (YOLO). In scenarios involving a maximum of 2 leakages, the outputs $\hat{y}_1$ to $\hat{y}_4$ represent the coordinates (x, y) of the 2 leakages. Outputs $\hat{y}_5$ denote the "leakageness" score

The issue of initializing the second leakage coordinate as zero or any arbitrary number presents a unique challenge in typical machine learning environments. A zero output or label doesn't mean the corresponding sample gets ignored during training. On the contrary, the model will attempt to predict zero for these samples, and any deviation from zero will contribute to the loss. An alternative approach is to mask or ignore the loss during training of the model for missing outputs (coordinates of second leakage when there is only one leakage), applying a binary mask to the corresponding output to disregard the loss contribution for missing outputs.

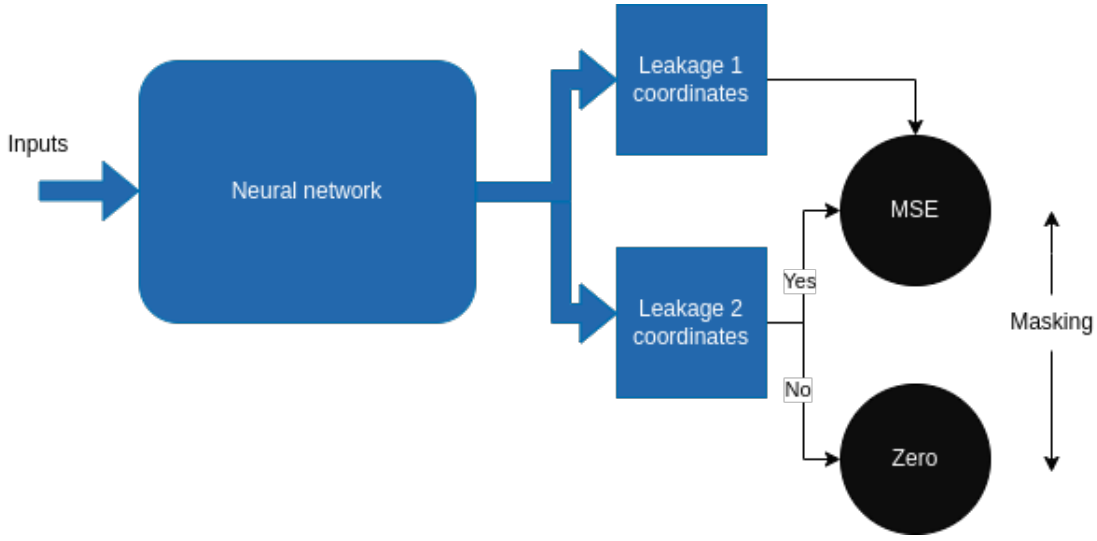


Figure 3.8: Masking on the computation of loss - In cases where the second leakage is absent, the loss calculated on the second leakage coordinate is set to zero. However, for samples where both leakages are present, the mean square error is computed as usual.

To address the issue of excluding specific samples (masking) from the loss computation, sample weights are commonly employed. However, a more direct approach involves managing the masking within the loss function itself. By creating a custom loss function, we evaluate the mean squared error (MSE) between the predicted and true values, specifically for instances where the second leakage exists as shown in Figure 3.8. This tailored loss function, named "masked_mse," calculates the MSE while disregarding contributions from the coordinates of the second leakage when it's absent. We incorporate this custom loss function during the compilation phase.

Chapter 4

Results & discussions

In the previous chapter, we outlined the process of configuring our model, which is designed to predict and pinpoint the locations of leakages within extensive industrial vacuum systems. This involved constructing a fully connected neural network and employing several strategies for target prediction. In the current chapter, we delve into the outcomes yielded by various numerical experiments using the diverse strategies employed for constructing these models. Moreover, we provide potential explanations for the observed results. All models were trained on a GeForce GTX 1650 Ti GPU using TensorFlow 2.12.1 [17]. Throughout, we used the MSE (leakage coordinates) and binary cross entropy loss ("leakageness" score), the Nadam optimizer [23] with standard parameters and trained for 1000 epochs with early stopping.

4.1 Case I: Single leakage

It becomes evident from both the data acquisition phase and the subsequent data analysis that an extremely rigorous data sanity check and cleaning process was imperative to develop a robust model. Consequently, we constructed several models employing varying degrees of data cleaning benchmarks and feature selections. Notably, enhanced feature engineering and more thorough data cleansing were noticeably linked to an enhancement in the accuracy score as shown in Table 4.1.

Mean absolute error (MAE) on dataset				
Model No.	**Model description**	**Train**	**Validation**	**Test**
Model1	Convex combination	0.0978	0.0790	0.0927
Model2	Linear regression model	0.1230	0.1218	0.1184
Model3	NN with full features	0.0913	0.1310	0.1527
Model4	NN with truncated features	0.0446	0.0796	0.0879
Model5	NN with truncated features and data cleansing (best)	0.0127	0.0737	0.0691

Table 4.1: Single leakage model - A comparison of the average mean absolute error (MAE) on the standardized training data, validation data, and test data is performed across various implemented models. These metrics are averaged over 10 iterations of each experiment. Detailed description of all the models discussed in the table are available in subsection 4.1.1

4.1.1 Machine learning experimentation

It took various iteration and experimentation to arrive at the best final model.

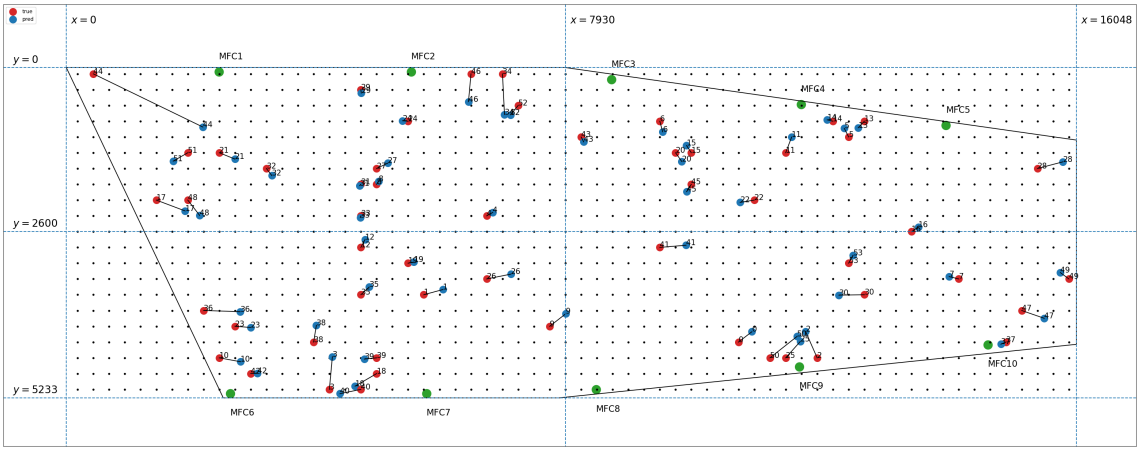


Figure 4.1: The image depicts the wing section of Airbus A350. Predicted and true leakage location on the vacuum bag based on the *best model*. True value (red) and its corresponding prediction (blue) are connected by a line.

- Model1: The Model1 corresponds to the convex combination method.

Despite its limitation, this approach provided respectable results with computational efficiency.

- Model2: This represents a basic linear regression model.
- Model3: This refers to the neural network (NN) trained before feature truncation
- Model4: This corresponds to the neural network (NN) trained after truncating features. Feature truncation was guided by feature importance scores estimated using importance analysis, resulting in the retention of only the 10 mass flow rates as the most important inputs for the optimal model as explained in section 3.2.1. The data cleansing process are outlined in the section 3.2.
- Model5: This represents the best neural network (NN) model trained with truncated features and an extensive data cleansing process. The best model (Model5) predictions and the true coordinates as plotted on the wing section is shown in Figure 4.1.

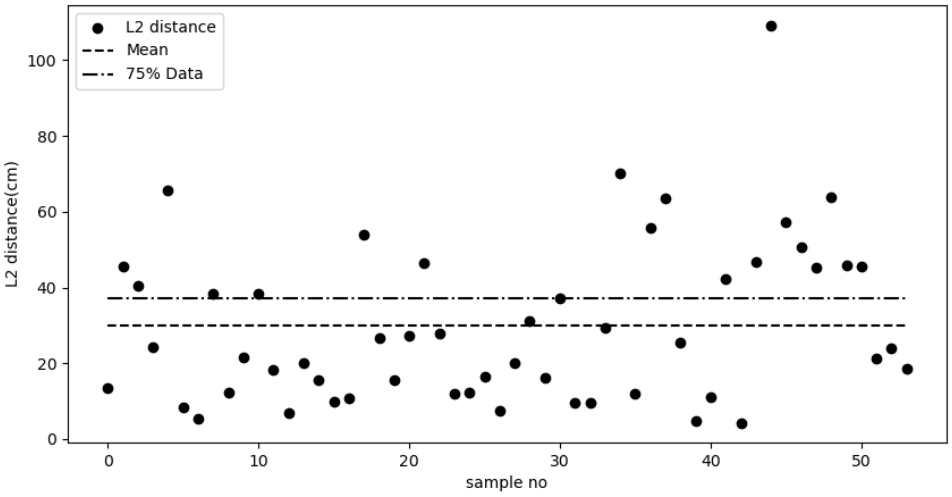


Figure 4.2: This plot displays the L2 distance between model leakage predictions and true values on each sample in the test data. This distance is a measure of model accuracy. It reveals that over 60% of the data points fall below the average distance of 30 cm. Furthermore, the lowest 75% exhibit a distance of less than 37 cm between true and model predictions.

4.1.2 Model evaluation

As depicted in Figure 4.2, it's evident that for the majority of the test data samples, the maximum error remains under 30 cm. This signifies a substantial advancement in comparison to existing methods. However, there is a noticeable disparity in predicting the leakage located in the upper left corner as shown in Figure 4.1. This divergence might stem from the asymmetrical nature of the geometry of the wing section. To address this and enhancing accuracy could involve accumulating more data from that specific corner and subsequently retraining the model, or assigning higher weighting to the data in that region during training. Unlike the potential flow model [1], the prediction error doesn't increase near the boundaries or in proximity to the vacuum ports.

It is possible that the trained model's performance could be constrained by the available resources and the time allocated for hyperparameter tuning. This limitation may indeed offer an opportunity for enhancing the model's accuracy with more extensive hyperparameter optimization.

4.1.3 Network architecture

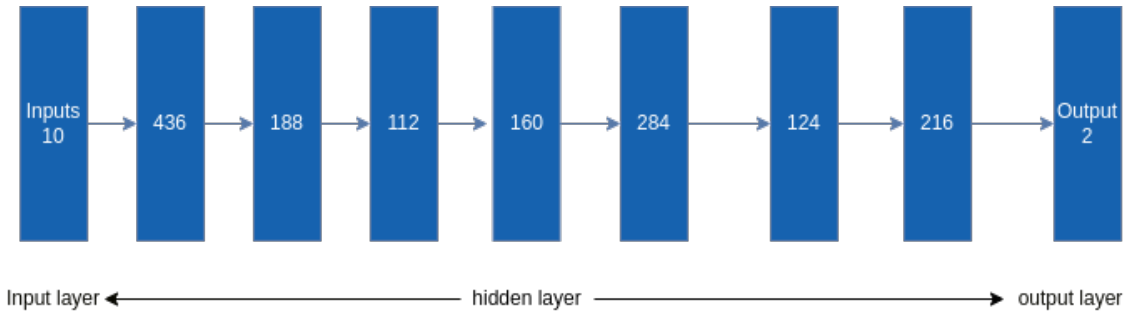


Figure 4.3: The architecture of the single-Leak neural network, selected through hyperparameter tuning. The rectangular box represents each layer. The numbers in each rectangular box represents the number of neurons in that layer.

The network architecture of the final model which gives best predictions on the test data is explained here. The fully connected network comprised 10 neurons in the input layer and 2 neurons in the output layer as shown in Figure 4.3. The RELU activation function was employed across all hidden layers, while the output layer utilized the linear activation function.

The model utilized the mean squared error loss function and the Nadam optimizer. Training was conducted using a batch size of 32, spanning 1000 epochs. Notably, early stopping and L1/L2 regularization were integrated to prevent overfitting.

4.1.4 The uncertainty of the model predictions

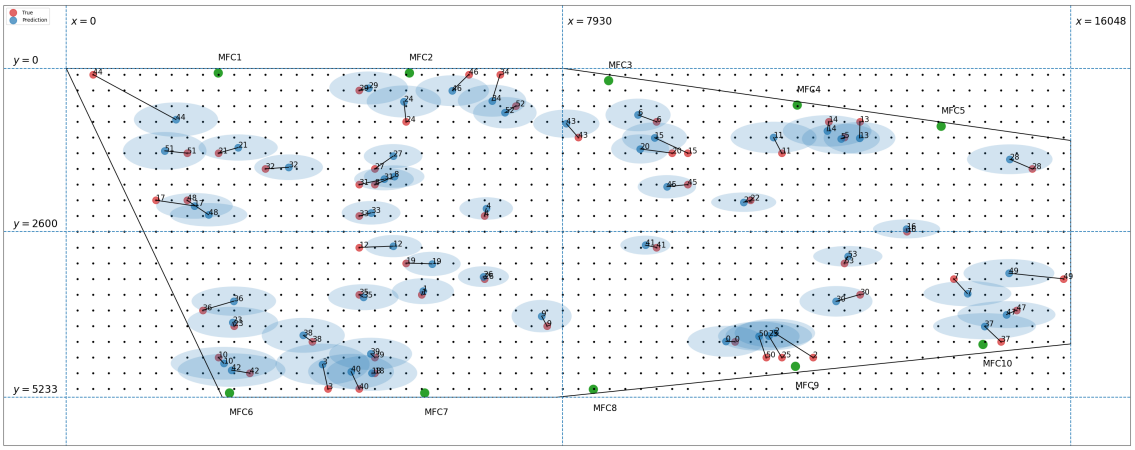


Figure 4.4: The image depicts the wing section of Airbus A350. Prediction uncertainty is assessed using the dropout method [6] on test dataset. The blue elliptical shape illustrates the region of confidence, with its center denoting the mean of over 1000 predictions. Unfortunately, for the 13 samples out of the 57 samples in the test data, the true value lies outside the region of confidence.

To estimate model uncertainty, the dropout method [6] is employed, as illustrated in Figure 4.4. Evidently, the uncertainty in the optimal model remains relatively low. It's worth mentioning that the model's uncertainty appears to be not as affected by leakage's location as much as that of the potential flow model [1]. Similar to the VFR Matching method, the prediction quality is relatively maintained even when the leakage position is situated in close proximity to the boundary, as discussed in [1]. Specifically, the percentage error in model prediction of the (x, y) coordinates is shown in Figure 4.5.

By employing dropout-based uncertainty quantification, we assessed the robustness of our model concerning dropout. In this context, a narrow region of uncertainty implies model robustness. Considering that the ma-

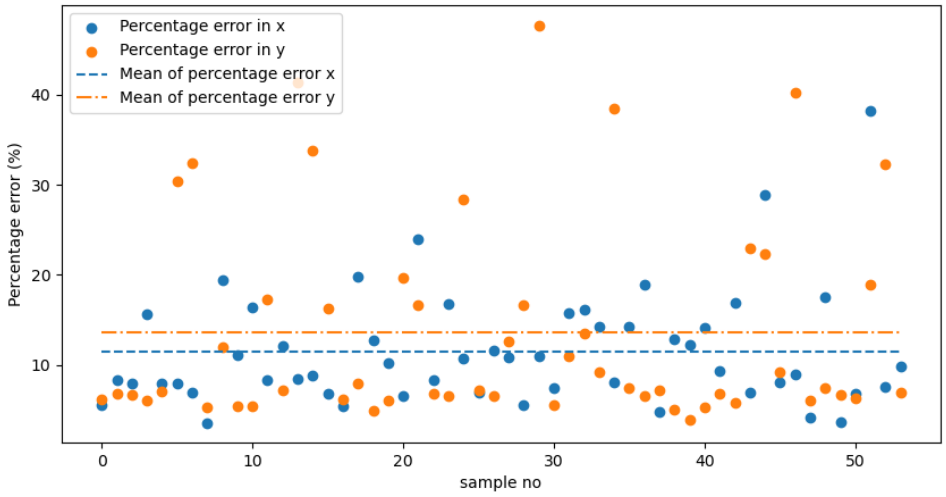


Figure 4.5: Prediction error in model predictions over each sample in test data is assessed using the dropout method [6]. This prediction error is estimated over 1000 predictions for each sample in test data. The mean percentage prediction error in (x, y) coordinates over all samples in the test data are $(11.5\%, 13.67\%)$ of the actual dimensions.

jority of predictions fall within the uncertain region (coloured with blue), it can be inferred that our model is not only robust but also accurate.

4.2 Case II: Multiple leakage

For multi-leakage case, we experimented with various ideas to arrive at the best final model. Detailed results of numerical experimentation are presented in Table 4.2.

4.2.1 Machine learning experimentation

The attainment of the optimal final model involved numerous iterations and experimentation, a process elaborated upon in this section.

- Model1: We initiated our experimentation with a straightforward linear regression (Model1) approach, aiming to predict four target variables representing the (x, y) coordinates of the leakages. The contrast in performance between Model1 and Model1.1 clearly

Mean absolute error (MAE) on dataset				
Model No.	**Model description**	**Train**	**Validation**	**Test**
Model1	Linear regression with randomly sorted leakages	0.6913	0.6115	0.6764
Model1.1	Linear regression with sorted leakages	0.3738	0.3392	0.3756
Model2	NN with randomly sorted leakages	0.1127	0.3234	0.5438
Model2.1	NN with sorted leakages	0.0178	0.1280	0.1343
Model3	NN Multi task learning	0.0257	0.1125	0.1126
Model4	NN Masked loss	0.0326	0.1113	0.1186
Model5	NN Multi task learning and Masked loss (best)	0.0075	0.0793	0.0836

Table 4.2: Multi leakage model - A comparison of the average mean absolute error (MAE) on the standardized training data, validation data, and test data is performed across various implemented models. These metrics are averaged over 10 iterations of each experiment. Detailed description of all the models discussed in the table are available in subsection 4.2.1.

demonstrates that sorting the model's outputs does indeed enhance its performance.

- Model2: The second model (Model2) introduced a regression neural network with 10 inputs and 4 outputs. Model2.1 improved accuracy by incorporating a technique where outputs were sorted during training, placing leakages with lower x-coordinates (sorting of coordinates) in the first two neurons.
- Model3: Subsequently, we delved into multi-task learning (Model3) artificial neural network (NN), extending the model to include the "leakageness" score as an additional output alongside the 4 coordinate outputs. Remarkably, all models employing multi-task learning were able to accurately predict the "leakageness" score, indicating the number of leakages, with perfect accuracy.
- Model4: Model4 (not multi-task learning), a artificial neural network (NN) maintained 10 inputs and 4 outputs, representing the

(x, y) coordinates of the two leakages, and introduced masking as an enhancement to Model2 and the model performed better than Model3.

- Model5: Model5, another artificial neural network (NN) is explored with the application of masking to the multi-task learning model (Model3) to assess its impact on accuracy. Predictions on the test data using the best model (Model5) plotted on the wing section is shown in Figure 4.6.

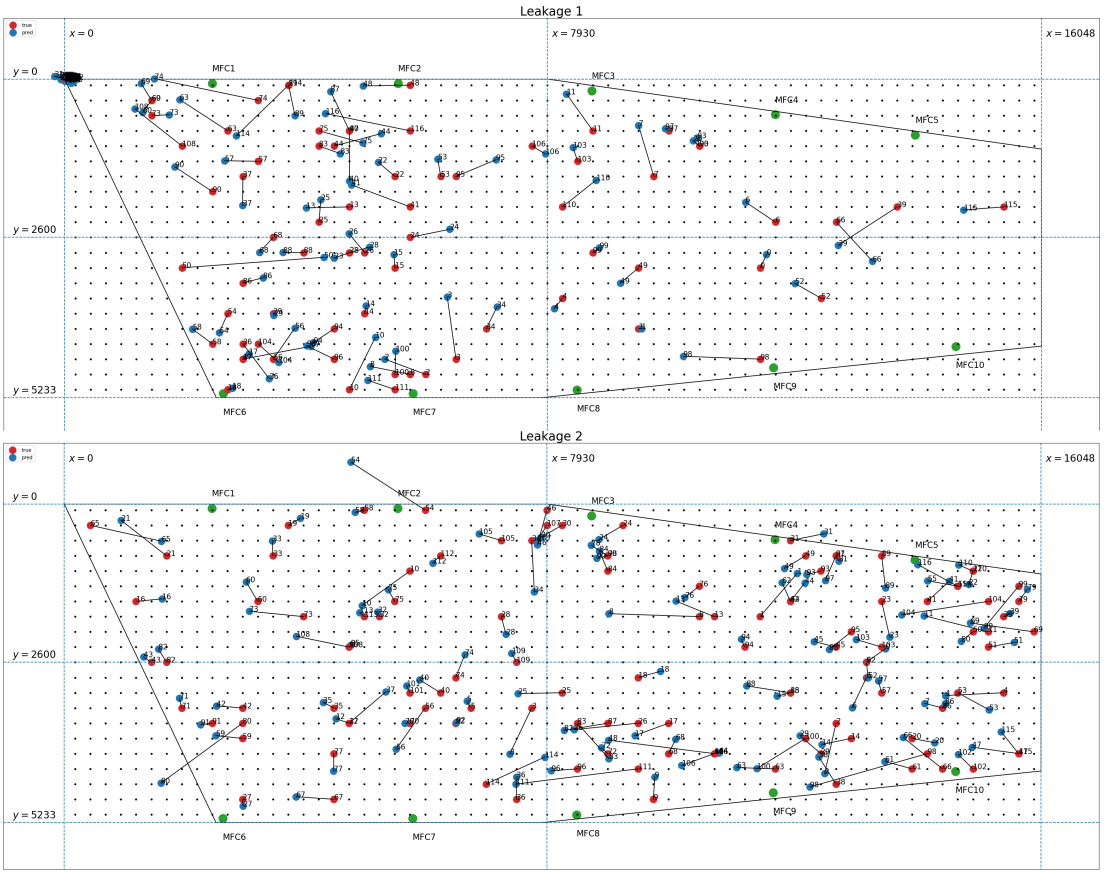


Figure 4.6: The image showcases the wing section of an Airbus A350, featuring two plots. The top plot illustrates Leakage 1, while the bottom plot displays Leakage 2 for the corresponding samples in the test dataset. Predicted coordinates (blue) and true coordinates (red) for leakages are connected by a line in each plot. In the case of samples with only one leakage present, it is plotted around $(0, 0)$ in the top plot.

4.2.2 Model evaluation

Dealing with multiple leakages is clealy more complex than handling a single leakage scenario. Distance between model leakage predictions and true values for both leakage 1 and leakage 2 are shown in Figure 4.7. The mean absolute error between the true and predicted value in the test data is notably higher for the multiple leakage case compared to the single leakage case.

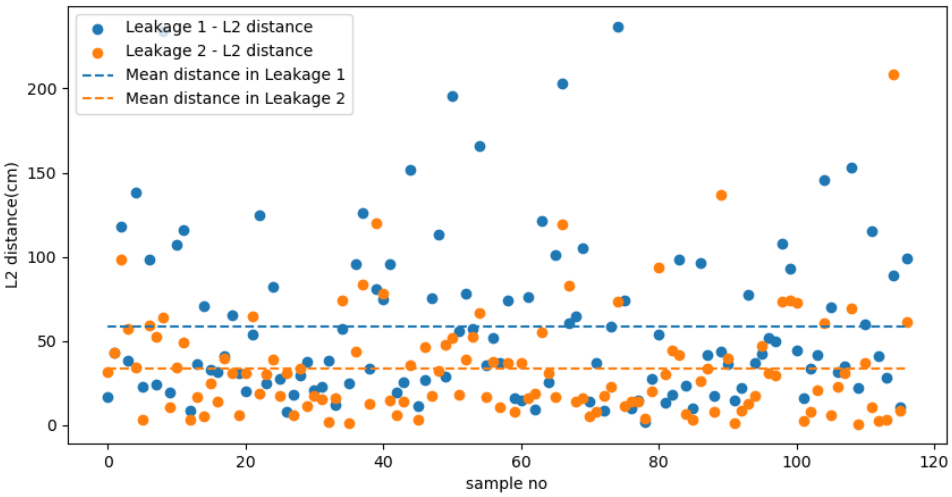


Figure 4.7: This plot displays the L2 distance between model leakage predictions and true values on each sample in the test data. This distance is measure of model accuracy. The distances for both leakage 1 and leakage 2 are separately plotted. Clearly, the mean distance between true and predicted leakage coordinates are higher in case of multi leakage model compared to single leakage model. In the multi leakage model, leakage 1 seems to have predicted with higher error.

Architecting a framework that adeptly shares information among tasks while minimizing interference poses a challenge in multi-task learning, especially when confronted with substantial differences in task complexity. In instances where tasks are relatively simple and share akin data distributions, training a multi-task model tends to be more straightforward. However, the complexity amplifies, and data characteristics diverge significantly in highly intricate tasks, rendering the endeavor more demanding [19].

A pivotal aspect of model training in this context is the weight assigned to the loss function of each task. This parameter emerged as critical during the iterative process of experimentation and model refinement. With two distinct tasks in play, the model incorporated two different loss functions, with the weighting of each task treated as a hyperparameter. Initially, assigning minimal weights to both losses yielded subpar accuracy in predictions. Adjustments were made to establish a specific relationship between the weights, ensuring that the more challenging task (identified through experimentation as the coordinate prediction task) received the highest weight.

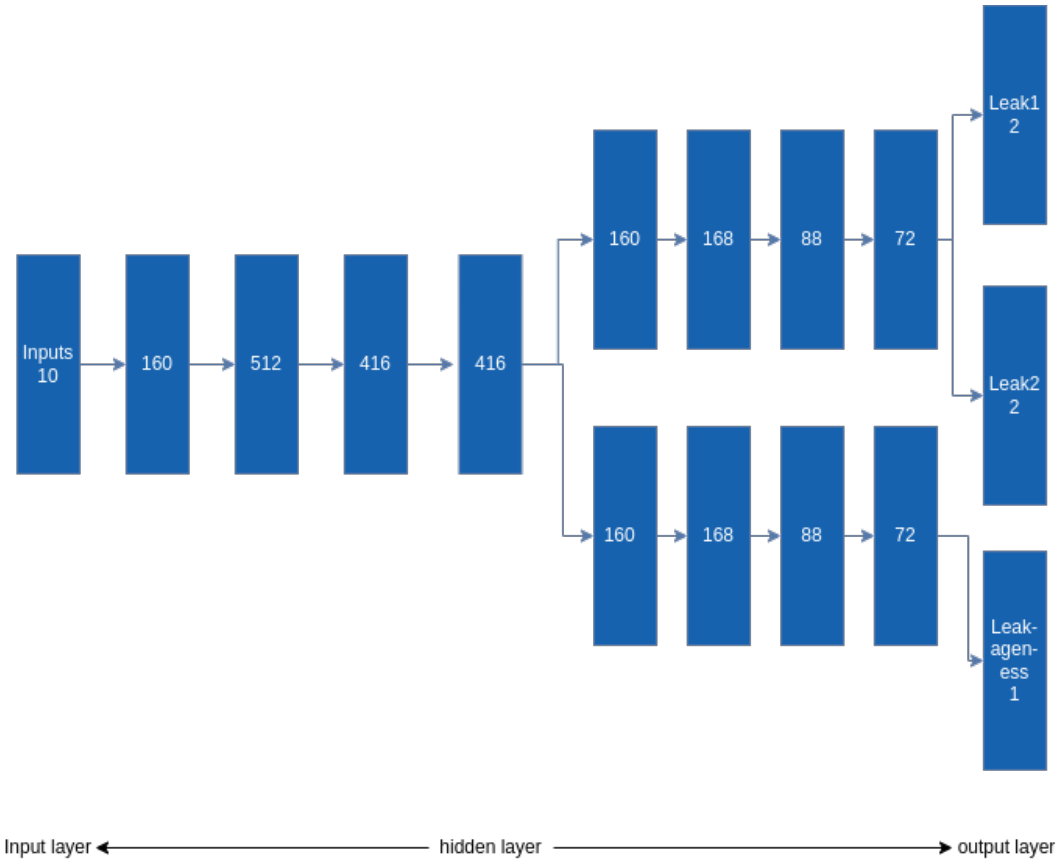


Figure 4.8: The architecture of the multi-leak neural network, selected through hyperparameter tuning. The rectangular box represents each layer. The numbers in each rectangular box represents the number of neurons in that layer. "Leak1" and "Leak2" represents the layer that prediction the (x, y) coordinates of the leakage 1 and leakage 2 respectively. The layer "Leakageness" predicts the "leakageness" score.

Despite these adjustments, the model initially fell short of surpassing the performance of Model2, a simple regression neural network with four outputs representing the (x, y) coordinates of two leakages. After numerous iterations and experiments, the multi-task learning neural network, equipped with masking and a fixed weight distribution of 0.9 for the leakage location task loss and 0.1 for the "leakageness" task loss, achieved optimal performance. It's worth noting that multi-task learning often necessitates intricate hyperparameter tuning, leaving room for potential further enhancement.

4.2.3 Network architecture

The network architecture of the final model which gives best predictions on the test data is explained here. The fully connected neural network was structured with 10 neurons in the input layer and 5 neurons in the output layer (4 for leakage coordinates and 1 for "leakageness"). A masked loss function was implemented to handle samples with a single leakage. The hidden layers used the Rectified Linear Unit (RELU) activation function, while the output layer employed the linear activation function. The model utilized the mean squared error loss function and the Nadam optimizer. Training was performed with a batch size of 32 over 1000 epochs, with the inclusion of early stopping and L1/L2 regularization to mitigate overfitting.

Chapter 5

Conclusion & future directions

In this project, we introduced a fully connected neural network architecture designed to predict leakage locations, addressing both single and multiple leakage cases. Through hyperparameter tuning, we successfully trained models with relatively high accuracy for both scenarios. While our models performed well, there were challenges, particularly in accurately predicting leakage locations in the top left corner of the airbus A350 wing section. We conducted numerous numerical experiments with diverse techniques for each scenario, progressively enhancing the model's accuracy. We also quantified the uncertainty of model predictions using the dropout method and visualized confidence domain for each sample in the test dataset. Additionally, we designed a neural network capable of predicting cases with multiple leakages with a considerable level of accuracy. This was achieved by leveraging concepts such as multi-task learning and a masking loss function.

To improve model accuracy, there is room for further study and opportunities. In the single leakage case, incorporating prior knowledge into the neural network to correctly project predicted values into the relevant voronoi cell offers a promising avenue for enhancement. Additionally, exploring scenarios in real industrial settings where more than two leakages may occur simultaneously, each with different diameters, presents an intriguing opportunity for further research.

Chapter 6

Appendix A

Model Deployment & Web App

As a extension to this student specialization project, this model is containerized and deployed. Then a containerized python-based web application that allows to input data and receive model predictions for the leakages in a vacuum bags as shown in Figure 6.1 and Figure 6.2. They are to be deployed on Nimbus, a modern cloud-compatible platform by German Aerospace Center (DLR) to transform awesome science into usable services.

Code

1. Source code and plots of the different strategies used are available in the GitHub link provided below:
   https://github.com/sreeragvn/leakage_detection_cfrp
2. Source code for the leakage detection app are available in the GitHub link provided below:
   https://github.com/sreeragvn/DLR-Leakage-Detection-Web-App

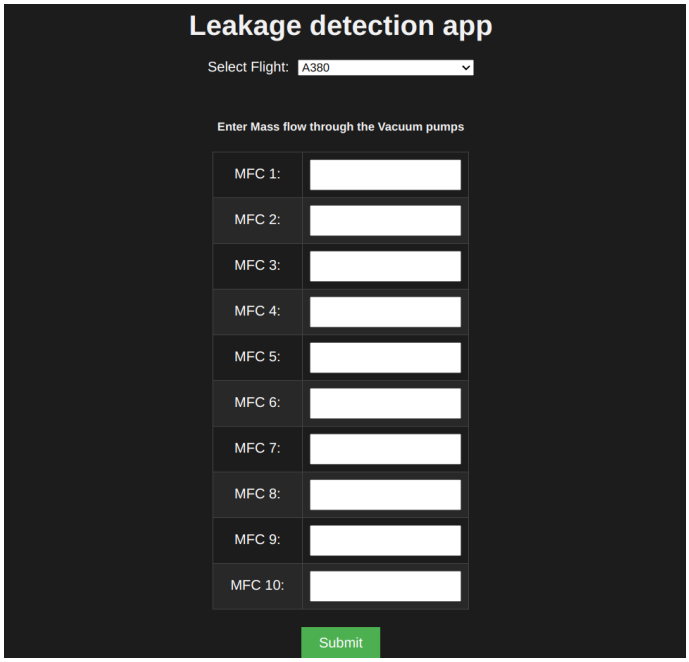


Figure 6.1: On the welcome page of the leakage detection web app, users can select the aircraft model (e.g. Airbus A320, A380) to specify the context for the prediction. Depending on the chosen aircraft model, the app will communicate with the corresponding model hosted on TensorFlow Serving docker for accurate predictions.

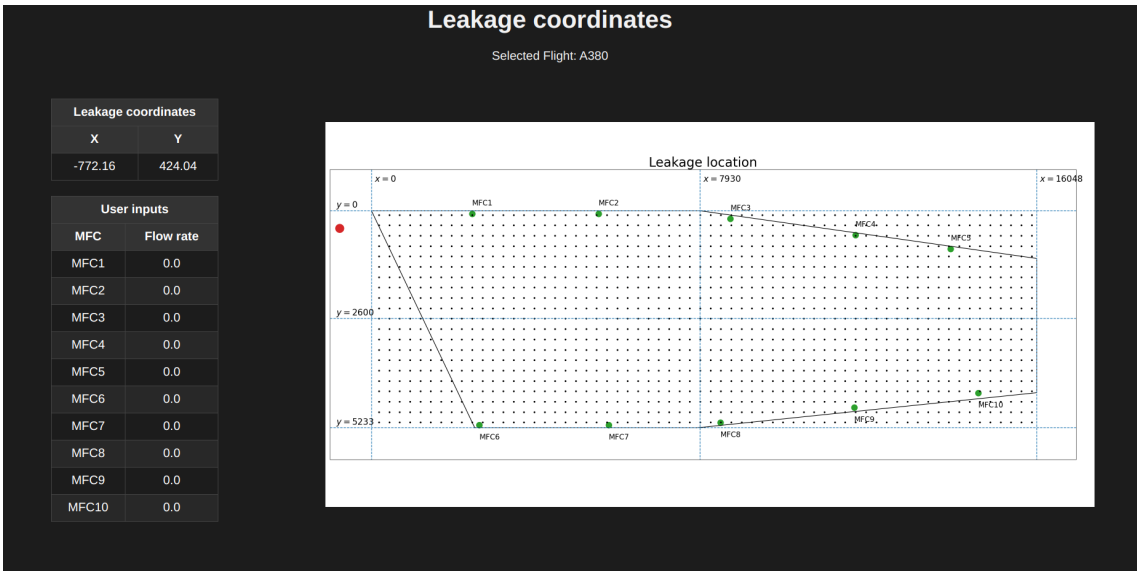


Figure 6.2: On the result page of the leakage detection web app, users can easily view the predicted leakage coordinates for their selected aircraft model.

Bibliography

[1] A. I. Haschenburger et al. "Computational methods for leakage localisation in a vacuum bag using volumetric flow rate measurements". In: *Production Engineering* 16.6 (2022), pp. 823–835. DOI: 10.1007/s11740-022-01129-8.

[2] Anja Haschenburger, Niklas Menke, and Jan Stüve. "Sensor-based leakage detection in vacuum bagging". In: *The International Journal of Advanced Manufacturing Technology* 116.7–8 (2021), pp. 2413–2424. DOI: 10.1007/s00170-021-07505-5.

[3] A. Haschenburger and C. Heim. "Two-stage leak detection in vacuum bags for the production of fibre-reinforced composite components". In: *CEAS Aeronautical Journal* 10.3 (2019), pp. 885–892. DOI: 10.1007/s13272-018-00357-y.

[4] Christoph Brauer, Dirk Lorenz, and Lionel Tondji. "Group equivariant networks for leakage detection in vacuum bagging". In: *2022 30th European Signal Processing Conference (EUSIPCO)*. 2022, pp. 1437–1441. DOI: 10.23919/EUSIPCO55093.2022.9909715.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2016. arXiv: 1506.02142 [stat.ML].

[7] Luis Oala et al. *Interval Neural Networks: Uncertainty Scores*. 2020. arXiv: 2003.11566 [cs.LG].

[8] Battlesnake. *Battlesnake/neural: LATEX: TikZ package for drawing neural networks. also available on CTAN at http://www.ctan.org/tex-archive/graphics/pgf/contrib/neuralnetwork*. URL: https://github.com/battlesnake/neural/tree/master.

[9] Vinod Nair and Geoffrey E Hinton. *Rectified linear units improve restricted Boltzmann machines*. URL: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[11] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[13] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer New York, 2016.

[14] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[15] Feiping Nie, Zhanxuan Hu, and Xuelong Li. "An investigation for loss functions widely used in machine learning". In: *Communications in Information and Systems* 18.1 (2018), pp. 37–52. DOI: 10.4310/cis.2018.v18.n1.a2.

[16] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[17] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[18] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html.

[19] Rich Caruana. In: *Machine Learning* 28.1 (1997), pp. 41–75. DOI: 10.1023/a:1007379606734.

[20] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[21] Jason Brownlee. *How to calculate feature importance with python*. Aug. 2020. URL: https://machinelearningmastery.com/calculate-feature-importance-with-python/.

[22] July 2023. URL: https://en.wikipedia.org/wiki/Convex_combination.

[23] Timothy Dozat. "Incorporating Nesterov Momentum into Adam". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016, pp. 1–4.