Politecnico di Torino

Avio Aero
a GE Aerospace company

Politecnico di Torino

Master's Degree in Aerospace Engineering

Graduation Session July 2026

APPLICATION OF AI FOR AUTOMATED MODAL SHAPE RECOGNITION METHODS

Supervisor

Prof. Daniele Botto

Candidate

Benedetto Morello

Tutors

Ing. Antonio D'Ettole
Ing. Giacomo David

Academic Year 2025–2026

Acknowledgements

Desidero ringraziare innanzitutto la mia ragazza e la mia famiglia, che mi hanno sempre sostenuto e accompagnato lungo questo cammino, cercando anche di capire cosa stessi studiando... pur senza riuscirci fino in fondo, visto che un bladed disk è stato scambiato persino per un palloncino colorato!!!
Un pensiero speciale a chi non ha potuto essere presente. A loro va il mio grato saluto.
Desidero infine ringraziare i miei tutor, Giacomo e Antonio, e il Professor Daniele Botto, per avermi guidato sempre nella direzione giusta e per aver reso questa esperienza non solo formativa, ma anche profondamente stimolante.

List of Figures

List of Tables

Contents

1 - Summary

Aeroelastic analyses of **bladed disks** are routinely performed using **Finite Element Methods (FEM)** to evaluate the dynamic behavior of aircraft engine components. These analyses produce complex-valued eigenvectors, which can be transformed into real-domain mode shapes for engineering interpretation. A major post-processing challenge is the classification of the large number of modes represented in Frequency vs Nodal Diameter **(FreND)** diagrams, a task that is currently performed manually by domain experts and is therefore time-consuming and difficult to scale. This thesis investigates the use of **Deep Learning** for **automated modal classification**. In particular, a **Convolutional Neural Network (CNN)** is designed to identify modal shape labels from processed aeroelastic data and to emulate the expert labeling procedure. The objective is not merely to reduce analysis time, but also to assess whether data-driven classification can achieve sufficient accuracy, robustness, and consistency to support engineering decision-making. The proposed approach is evaluated on a real dataset of turbine blade modes provided within an industrial collaboration with **GE Avio Aero** and discussed in terms of predictive performance, generalization capability, and applicability to industrial workflows.

2 - Introduction

2.1 Problem Outline

This thesis investigates an **artificial-intelligence-based framework** developed to eliminate human involvement in the **labeling** of **blade modal shapes**. This process is currently time-consuming and highly error-prone; therefore, the development of an AI-based tool capable of performing this task has become necessary for aeroelastic analyses aimed at investigating blade vibrational characteristics and interactions with the fluid.
The first step of the project was to identify the nature of the problem itself, namely the **classification** of blade modal shapes. The blade modal shapes derived from **Finite Element Analyses** (FEA) are represented in the form of complex-valued matrices, which contain the complex displacement of each node of the FEM model representing a shape. One of the challenges encountered in this work stems from the fact that this mathematical representation is not unique. Indeed, the same label can correspond to different matrices. This property is inherent to the time-varying nature of the eigenmode, whose evolution is periodic in time.
At the beginning, the idea was to adopt a conventional **Machine Learning** (ML) approach, in which the AI model maps a set of physically meaningful input features to a unique output label. However, it soon became evident that, because of the very large number of parameters characterizing blade modal shapes, as well as the intrinsic complexity of the problem, a **Deep Learning** (DL) approach would be more suitable. Unlike traditional machine learning methods, Deep Learning enables the model to automatically extract and construct the relevant features during the training phase, thus making it particularly effective for high-dimensional and highly nonlinear classification tasks.
Another key insight that significantly simplified the problem was the introduction of a preprocessing transformation aimed at converting the extremely intricate blade geometries and their corresponding modal shape patterns into colored simplified maps. As a result, the original task was reformulated as an image recognition problem.
The overall objective is not fundamentally different from that of many AI-based systems currently employed in a wide range of fields, including medicine, security, and everyday consumer applications. In all these domains, artificial intelligence has achieved remarkably high levels of accuracy, enabling results that remain impressive even today. Nevertheless, the framework developed in this work is distinguished by its **physics-based** nature. This feature ensures that the recognition of blade modal shapes is not only accurate, but also fully consistent with the governing physical equations underlying the phenomenon.

2.2 Predictive Models

To improve the results, it is necessary to clearly define the task we expect the code to perform. For this reason, it is worth taking some time to discuss the different categories of models.
Predictive modeling comprises a range of techniques aimed at estimating dependent variables based on sets of inputs, commonly referred to as predictors. In this context, artificial intelligence can be understood as a collection of techniques capable of predicting outcomes and generating responses to previously unseen problems by learning underlying patterns from data. These techniques can generally be divided into two main categories:

- **Classification** problems involve processing input data such as patterns in the form of matrices or images and assigning one or more labels;
- **Regression** problems focus on reconstructing an underlying functional relationship in order to generate quantitative predictions of an output, thereby avoiding onerous analyses.

The objectives of this thesis fall squarely within the scope of a classification problem.

2.2.1 Classification Scope

Classification can be approached through different types of algorithms, depending on the objective of the task. First, it is useful to distinguish between the concepts of class and label. A **class** refers to the mutually exclusive category, typically used in problems where only one object, or one dominant object, is expected in the image. A **label**, instead, is defined as an output that can coexist with other labels.
With this distinction in mind, the main types of classification can be summarized as follows:

- **Binary** classification is the simplest form of classification. Its purpose is to determine whether a specific item, condition, or feature is present or absent. This approach is widely used, for example, in medical diagnosis and security applications.
- **Multi-class** classification refers to tasks in which the model must assign an input to exactly one class among several possible classes. This approach is commonly used in applications such as image recognition.
- **Multi-label** classification is used when multiple objects, attributes, or categories can coexist in the same input. In this case, the model assigns one or more labels simultaneously. This approach is often adopted in image tagging and topic assignment tasks.
- **Multi-task** classification extends these ideas by solving multiple related tasks at the same time. For example, a model may first identify an object and then provide additional information about it, such as its properties or condition.

For the purposes of this thesis, a multi-label classification approach is the most appropriate.

2.3 Machine Learning vs Deep Learning

Both Machine Learning (ML) and Deep Learning (DL) belong to the broader field of Artificial Intelligence (AI), as shown in Fig.2.1; however, they differ significantly in terms of methodology and level of abstraction.

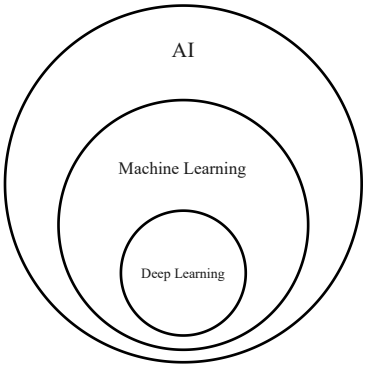


Figure 2.1: AI Venn diagram

In traditional **Machine Learning**, the model requires an input vector composed of selected variables, usually referred to as features, in order to perform a prediction or classification task. This approach demands a strong physical understanding of the phenomenon under investigation, since the relevant features must be identified, extracted, and correlated with the desired output. In this context, a preliminary classification task was explored for the modal shape labeling of a simple beam-like structure. By using quantities such as the number of modal nodes and energy-related parameters, it was possible to classify the modal shapes of a blade modeled as a beam.
Deep Learning, on the other hand, provides a higher degree of autonomy to the model. Instead of relying on manually engineered features, the network automatically extracts the most relevant representations directly from the input data, which are often provided in the form of vectors, matrices, or higher-dimensional tensors. Therefore, while Machine Learning depends on features explicitly selected by the researcher, Deep Learning is capable of learning such features automatically from the available data.
In summary, Machine Learning learns from manually defined input features, whereas Deep Learning learns both the features and the mapping to the output directly from the data.

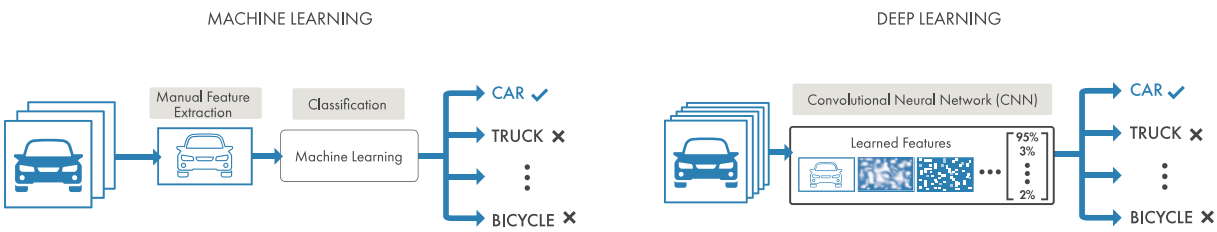


Figure 2.2: ML vs DL[5].

Learning can be divided into three categories:

- **supervised**, if training is performed on labeled datasets, similarly to this thesis;
- **unsupervised**, in which the model learns patterns and structure from the data and is not strictly a classification approach;
- **reinforcement**, if the model learns through trial-and-error loops, particularly in robotics applications.

2.4 AI Overview

Neural networks usually take as input a dataset X composed of N samples. Each sample may consist of a vector, a matrix, an image (colors are represented by scalars) or, more generally, a tensor. To clarify the notation, the following symbols are adopted:

Symbol	Meaning
$Y_c^{(n)} \in \mathbb{B}^{1\times C}$	True numerical value associated with label represented by channel c for sample n, treated as correct based on the user's judgment.
$y^{(n)}$	True alphanumeric label string for sample n.
$\hat{Y}_c^{(n)} \in \mathbb{R}^{1\times C}$	Predicted numerical value associated with channel c for sample n, such as a score or probability produced by the model.
$\hat{y}^{(n)}$	Predicted label for sample n, obtained after post-processing the model outputs.

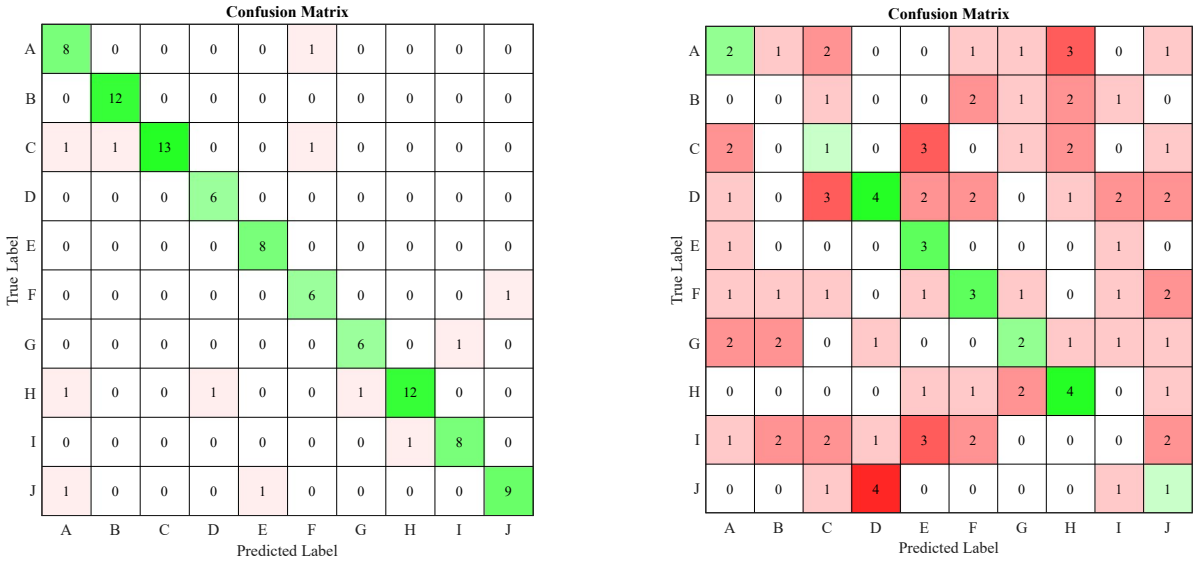
Table 2.1: Notation used.

2.4.1 Confusion Matrix

Performance refers to the ability of a model to correctly predict outcomes by generalizing the knowledge learned during the training phase. Several mathematical indicators can be used to evaluate model performance by comparing the true labels with the predicted class.

A first useful tool for performance evaluation is the **Confusion Matrix**, which is a table reporting the true classes on the ordinate axis and the predicted classes on the abscissa axis. For a highly accurate algorithm, most predicted classes coincide with the true classes; therefore, the majority of the values are concentrated along the main diagonal, as shown in Fig.2.3a.

Conversely, when the error rate is high, the values are more widely distributed outside the diagonal, and the diagonal cells tend to contain fewer elements, as shown in Fig.2.3b.



(a) (b)

Figure 2.3: Confusion matrices for (a) high and (b) low classification accuracy.

Now, let us define some quantities referring to a binary classification problem:

TP = True Positive, correct positive result;

FP = False Positive, wrong positive result;

FN = False Negative, wrong missing result;

TN = True Negative, correct missing result.

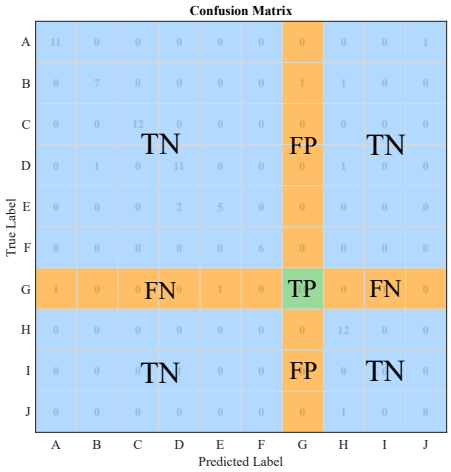


Figure 2.4: Binary Confusion Matrix.

2.4.2 Multi-label Classification Metrics

Metrics are parameters to compute the performance of a model, based on predictions and corresponding true results[8].
Accuracy is defined as the proportion of correct predictions over all samples. It can be expressed in two different forms. *Subset accuracy* is more restrictive, since it measures the proportion of samples for which the predicted label set exactly matches the true label set. By contrast, in the multilabel setting, *accuracy* is based on the intersection between predicted and true label sets, averaged over all samples:

$$\text{Subset Accuracy} \doteq \frac{1}{N}\sum_{n=1}^{N} I\left[y^{(n)} = \hat{y}^{(n)}\right] \tag{2.1}$$

$$\text{Accuracy} \doteq \frac{1}{N}\sum_{n=1}^{N} \frac{\left|\hat{y}^{(n)} \cap y^{(n)}\right|}{\left|\hat{y}^{(n)} \cup y^{(n)}\right|} \tag{2.2}$$

where $I[\cdot]$ denotes the Iverson bracket operator:

$$I[E] \in \mathbb{B} = \begin{cases} 1 & \text{if } E \text{ is true} \\ 0 & \text{if } E \text{ is false} \end{cases}$$

Hamming Score (HS) is related to the Hamming Loss (HL) and measures how well the model predicts each channel independently, averaging the result over all samples and channels:

$$\text{HL} \doteq \frac{1}{NL}\sum_{n=1}^{N}\sum_{c=1}^{C} I\left[Y_c^{(n)} \neq \hat{Y}_c^{(n)}\right] \qquad \Rightarrow \qquad \text{HS} \doteq 1 - \text{HL} \tag{2.3}$$

where L is the total number of labels.
Precision and **Recall** measure the proportion of correctly predicted labels with respect to the predicted and true label sets, respectively. In the example-based multilabel setting, they are defined as:

$$\textit{Precision} \doteq \frac{1}{N}\sum_{n=1}^{N} \frac{\left|y^{(n)} \cap \hat{y}^{(n)}\right|}{\left|\hat{y}^{(n)}\right|} \qquad \textit{Recall} \doteq \frac{1}{N}\sum_{n=1}^{N} \frac{\left|y^{(n)} \cap \hat{y}^{(n)}\right|}{\left|y^{(n)}\right|} \tag{2.4}$$

Thus, maximizing Precision reduces false positives, while maximizing Recall reduces false negatives. In practice, there is often a trade-off between the two metrics.
F_β**-score** is the weighted harmonic mean of Precision and Recall:

$$F_\beta \doteq \frac{(1+\beta^2)(Precision \cdot Recall)}{\beta^2 \cdot Precision + Recall} \tag{2.5}$$

The most commonly used case is the F_1**-score**, which assigns equal importance to Precision and Recall. In the example-based multilabel setting, it is defined as:

$$F_1 \doteq \frac{1}{N}\sum_{n=1}^{N}\frac{2\cdot\left|\hat{y}^{(n)}\cap y^{(n)}\right|}{|\hat{y}^{(n)}|+|y^{(n)}|} \tag{2.6}$$

Micro **−** **and** ***Macro*** **−** $\boldsymbol{F_1}$ **Scores** are commonly used in multi-label classification to aggregate performance across labels. Since the F_1 score has already been introduced, it is defined here in the same way as the harmonic mean of precision and recall; what changes is how these quantities are aggregated across classes.
In the Micro formulation, true positives, false positives, and false negatives are summed over all labels before computing precision and recall:

$$Micro-Precision \doteq \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c+FP_c)} \qquad Micro-Recall \doteq \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c+FN_c)} \tag{2.7}$$

The resulting $Micro-F_1$ is then computed from $Micro-Precision$ and $Micro-Recall$ using the standard definition.
In the Macro formulation, precision, recall, and F_1 are first computed separately for each label c, and then averaged:

$$Macro-Precision \doteq \frac{TP_c}{TP_c+FP_c} \qquad Macro-Recall \doteq \frac{TP_c}{TP_c+FN_c} \tag{2.8}$$

$$Macro-F_1 \doteq \frac{1}{C}\sum_{c=1}^{C} F_{1,c} \tag{2.9}$$

$Micro-F_1$ gives more weight to frequent labels because all decisions are pooled together, whereas $Macro-F_1$ assigns equal importance to each label. Therefore, $Macro-F_1$ is often more informative in imbalanced multi-label settings, as it better reflects performance on minority labels.

2.4.3 ROC

The Receiver Operating Characteristic (ROC) curve provides a general visualization of the classifier's ability to discriminate between positive and negative classes. In the multilabel setting, ROC analysis is performed by considering the equivalent binary formulation of the problem shown in Fig.2.4, where each sample-label pair is treated as an independent binary decision. The curve reports the true positive rate (TPR) on the ordinate axis and the false positive rate (FPR) on the abscissa axis, where:

$$\text{TPR} \doteq \frac{TP}{TP+FN} \qquad \text{FPR} \doteq \frac{FP}{FP+TN} \tag{2.10}$$

Fig.2.5 shows different model behaviors. An ideal classifier correctly predicts all positive label assignments and rejects all negative ones, hence $TPR = 1$ and $FPR = 0$. On the contrary, a random classifier cannot distinguish between positive and negative label assignments, so its response is determined by chance.

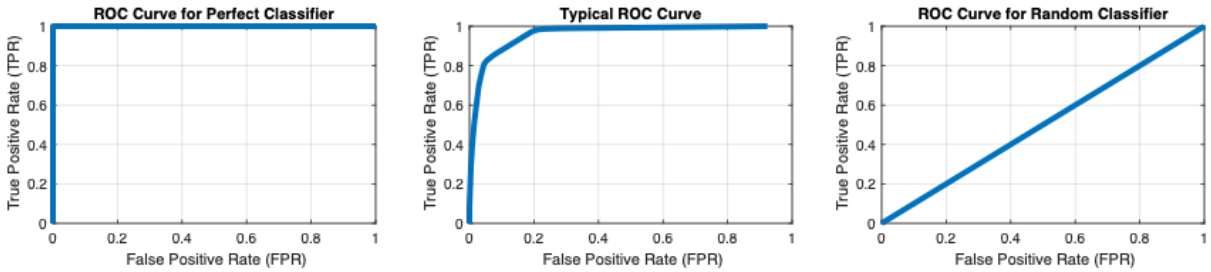


Figure 2.5: ROC curves[11].

2.4.4 Underfitting and Overfitting

Underfitting and overfitting are opposite phenomena that characterize the behavior of an algorithm when learning from a training dataset. In particular, overfitting occurs when the model adapts too closely to the training data, capturing also noise or specific fluctuations, and therefore fails to generalize well to unseen test data as shown in Fig.2.6. In contrast, underfitting occurs when the model is unable to capture sufficient information from the training data, resulting in poor performance even on the training set.

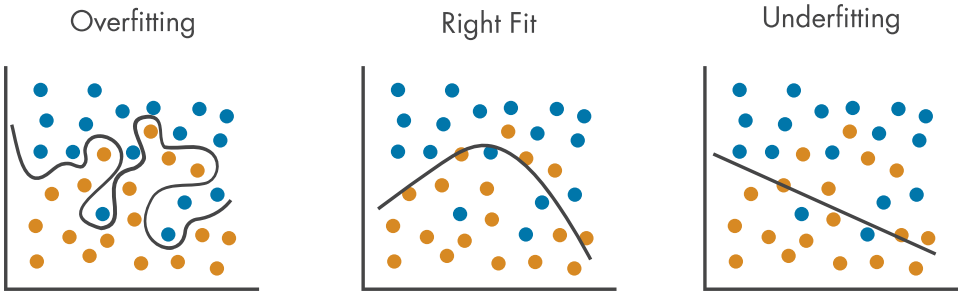


Figure 2.6: Illustration of Underfitting, Good Fit, and Overfitting[7].

2.4.5 Imbalanced Dataset Problem

A challenge frequently encountered in machine learning is the class imbalance problem, that occurs when the number of samples belonging to one class is significantly higher than the number of samples belonging to another. In such cases, a model may achieve apparently high overall accuracy while still performing poorly on the minority class, which is often the most relevant one in practical applications.
Several approaches have been proposed in the literature to address this problem[9]. These can be broadly divided into two categories: algorithm-level techniques and data-level techniques.
Algorithm-level approaches aim to improve the learning process without modifying the original dataset.
These methods include:

- The use of evaluation metrics that better capture classifier performance on imbalanced datasets, such as Matthews Correlation Coefficient (MCC) and AUC-ROC (Area Under the Receiver Operating Characteristic Curve);

- The adoption of classifiers that are generally more robust in the presence of class imbalance, such as Random Forest or AdaBoost;
- Anomaly detection methods, in which the minority class is treated as an anomaly;
- Cost-sensitive learning, where higher penalties are assigned to misclassifications involving the minority class.

Data-level techniques are more intuitive since they directly modify the training dataset in order to reduce class imbalance.
The main strategies are the following:

- **Oversampling** increases the number of minority-class samples until a more balanced distribution is achieved. A simple method is random oversampling, which replicates existing minority samples. However, this may lead to overfitting, as also observed during this thesis work. A more effective solution is the use of synthetic sample generation techniques, such as SMOTE (Synthetic Minority Over-sampling TEchnique) in Fig.2.7, which creates new artificial samples that remain faithful to the original data distribution. Other algorithms, such as ADASYN, are also available. The main drawback of oversampling is the increased computational cost during training.

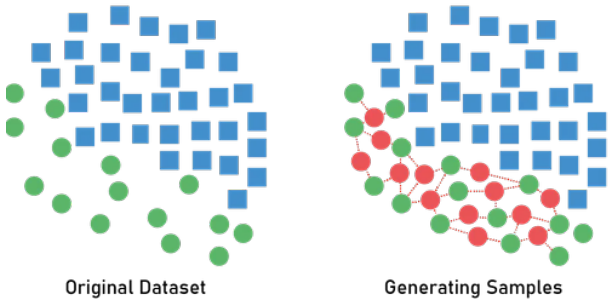


Figure 2.7: SMOTE interpolation technique[10].

- **Undersampling** reduces the number of majority-class samples until it matches, or more closely matches, the minority class as descibed in Fig.2.5. While this method decreases training time, it may discard useful information contained in the removed samples.
- **Combined resampling** combines oversampling and undersampling techniques in order to benefit from both approaches, balancing the dataset while trying to limit overfitting and information loss.

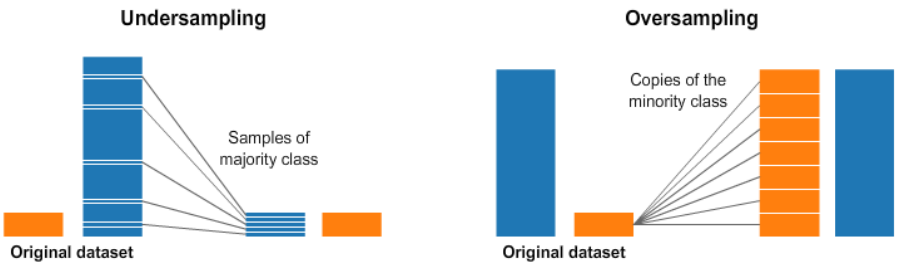


Figure 2.8: (Random) Oversampling and Undersampling[12].

3 - Mathematical Methods for Artificial Intelligence

3.1 Empirical Risk Minimization

With reference to Sec. 2.4, the training of a neural network is fundamentally based on reducing the loss, i.e., the discrepancy between the predicted outputs and the corresponding ground-truth labels, through iterative updates of the model parameters. More precisely, the learning process aims to approximate the unknown target function f by means of an approximated model $\hat{f}$, which maps an input $X^{(n)}$ to a score or probability distribution:

$$\hat{f}(X^{(n)}; W) = \hat{Y}^{(n)} \approx Y^{(n)} \tag{3.1}$$

where W denotes the trainable weights of the network. The final predicted label $\hat{y}^{(n)}$ is then obtained from $\hat{Y}^{(n)}$ through a thresholding.
More formally, let $\mathcal{D}$ [13] denote the data distribution over the true space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $X \in \mathcal{X}$ is the input and $Y \in \mathcal{Y}$ is the ground-truth score; therefore, each sample is defined as $Z^{(n)} = (X^{(n)}, Y^{(n)}) \in \mathcal{Z}$.
The population risk associated with the model parameters W is defined as the expected value of a loss function $\ell(Z; W)$ with respect to the random variable $Z \sim \mathcal{D}$, namely:

$$R(W) \doteq \mathbb{E}_{Z \sim \mathcal{D}} \left[\ell(Z; W) \right] \tag{3.2}$$

where ℓ can be written explicitly as $\ell(Y, \hat{Y})$, but given that $\hat{Y} = f(X; W)$, it can be more compactly expressed as $\ell(Z; W)$.
Since the distribution $\mathcal{D}$ is limitless, the population risk cannot be evaluated exactly over the whole space $\mathcal{Z}$. In other terms, to train a neural network it is necessary to consider a finite subset training set $\mathcal{T}$:

$$\mathcal{T} = \left\{ Z^{(1)}, Z^{(2)}, \ldots, Z^{(N)} \right\} \subset \mathcal{Z} \tag{3.3}$$

It is then necessary to generalize the knowledge gained from $\mathcal{T}$ by averaging over the samples, thereby defining the empirical risk:

$$R_{\mathcal{T}}(W) \doteq \frac{1}{N} \sum_{n=1}^{N} \ell\Big(Z^{(n)}; W \Big) \tag{3.4}$$

Accordingly, the Empirical Risk Minimization (ERM) problem can be formulated as:

$$\min_{W \in \mathcal{W}} R_{\mathcal{T}}(W) = \min_{W \in \mathcal{W}} \frac{1}{N} \sum_{n=1}^{N} \ell\Big(Z^{(n)}; W \Big) \tag{3.5}$$

where $\mathcal{W}$ denotes the space of admissible model parameters.

3.2 Loss Functions for Classification Problems

To define an appropriate numerical optimization procedure, it is essential to select a loss function that is consistent with the structure of the classification task. In supervised learning, a loss function quantifies the discrepancy between the target distribution $Y^{(n)}$ and the predicted distribution $\hat{Y}^{(n)} = \hat{f}(X^{(n)}; W)$ for a single sample n. By contrast, the cost function is the average of the loss values over the entire training dataset [18]. Accordingly, the optimization process seeks to minimize the empirical risk by reducing the loss on individual samples.
A suitable loss function could satisfy some analytical properties [19], such as:

- *Continuity*, a function g is continuous at a point a if:

$$\lim_{x \to a} g(x) = g(a) \tag{3.6}$$

- *Differentiability*, a function g is differentiable at a if:

$$g'(a) = \lim_{h \to 0} \frac{g(a+h) - g(a)}{h} \tag{3.7}$$

- *Lipschitz continuity*, a function g is Lipschitz continuous if there exists a constant $K \geq 0$ such that:

$$|g(x_1) - g(x_2)| \leq K \|x_1 - x_2\| \qquad \forall x_1, x_2 \in \operatorname{dom} g \tag{3.8}$$

  where K is called the Lipschitz constant.

- *Convexity*, a function $g : \mathbb{R}^n \to \mathbb{R}$ is convex if $\operatorname{dom} g$ is a convex set and:

$$g(\theta x_1 + (1-\theta)x_2) \leq \theta g(x_1) + (1-\theta) g(x_2) \qquad \forall x_1, x_2 \in \operatorname{dom} g,\ 0 \leq \theta \leq 1 \tag{3.9}$$

  Geometrically, this means that every chord joining two points of the graph lies above the graph itself. Moreover, g is strictly convex if the inequality is strict for $x_1 \neq x_2$ and $0 < \theta < 1$. A sufficient condition for strict convexity is that the Hessian exists and is positive definite. However, in deep learning applications the empirical risk is generally a non-convex function.

Classification loss functions can be broadly divided into two categories, as shown in Fig. 3.1.

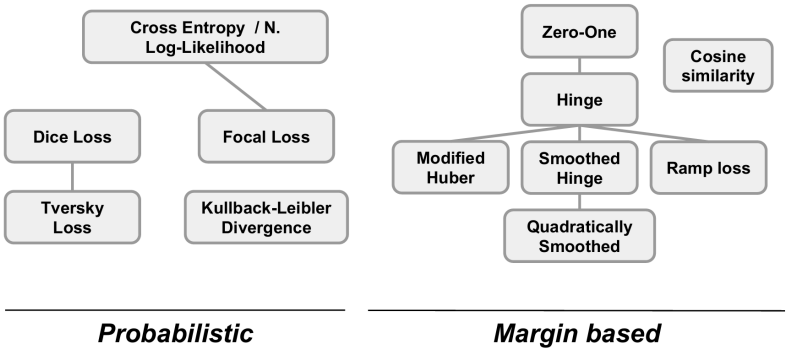


Figure 3.1: Groups of classification loss functions [19].

Probabilistic loss functions are designed to compare probability distributions over the labels; this type of loss quantifies how accurately $\hat{Y} \in [0,1]^{1\times C}$ reproduces the target $Y \in \mathbb{B}^{1\times C}$. By contrast, margin-based loss functions operate on scores $\hat{Y} \in \mathbb{R}^{1\times C}$ and seek to enlarge the separation margins by emphasizing one or more labels. Also, while in probabilistic loss Y is boolean (multi-hot encoding), in margin-based formulations the target is typically encoded as $Y \in \{-1,+1\}$. In other words, the choice of loss function depends on the meaning assigned to the neural network outputs, whether they represent probabilities or simply numerical values with no direct physical interpretation.

3.2.1 Hinge Loss family

Among margin-based functions, the Hinge (H) loss is widely employed[22]:

$$\ell^{(H)}(Z;W) = \frac{1}{C}\sum_{c=1}^{C} \max\left(0, 1 - \hat{Y}_c^{(n)} Y_c^{(n)}\right) \tag{3.10}$$

By averaging over the channels, the corresponding empirical risk is:

$$R_{\mathcal{T}}^{(H)}(W) = \frac{1}{N}\sum_{n=1}^{N} \ell^{(H)}(Z;W) = \frac{1}{NC}\sum_{n=1}^{N}\sum_{c=1}^{C} \max\left(0, 1 - \hat{Y}_c^{(n)} Y_c^{(n)}\right) \tag{3.11}$$

Note that the hinge loss is convex but not strictly convex, and it is not differentiable at the margin boundary. These properties may complicate the optimization process and typically require the use of sub-gradient methods. The hinge loss can be interpreted as a convex surrogate, and an upper bound, of the Zero-One (ZO) loss:

$$\ell^{(ZO)}(Z;W) = \frac{1}{C}\sum_{c=1}^{C} I\left[\hat{Y}_c^{(n)} Y_c^{(n)} < 0\right] \tag{3.12}$$

However, several variants have been proposed to address this characteristic, either by using sub-gradients or by smoothing the function. Finally, a more robust variant is the Ramp (R) loss, which consists of a truncated Hinge loss [19]:

$$\ell^{(R)}(Z;W) = \frac{1}{C}\sum_{c=1}^{C} \min\left(1, \max\left(0, 1 - \hat{Y}_c^{(n)} Y_c^{(n)}\right)\right) \tag{3.13}$$

It is non-convex; therefore, optimization is generally more challenging.

3.2.2 Cross-Entropy Loss family

Cross-entropy (CE) is one of the most widely used probabilistic loss functions in classification problems, especially in deep learning models for pattern recognition.
As a consequence, in the present work the cross-entropy is used because the modal shape recognition problem is formulated as a classification task in which the CNN assigns each input sample to one or more label.
In classification, the sample-wise cross-entropy measures the discrepancy between the target distribution $Y^{(n)}$ and the predicted one $\hat{Y}^{(n)}$ for the n-th sample and c-th channel:

$$\ell^{(CE)}(Z;W) = -\sum_{c=1}^{C} Y_c^{(n)} \ln\left(\hat{Y}_c^{(n)}\right) \tag{3.14}$$

Its expression is closely related to Shannon entropy, which measures the uncertainty associated with a probability distribution P over C channels[21]:

$$H(P) \doteq -K \sum_{c=1}^{C} P_c \ln(P_c) \quad [nats] \tag{3.15}$$

where K is a positive constant (in this context $K \equiv 1$).
The entropy quantifies the uncertainty of the distribution and reaches its maximum when the distribution is uniform, i.e. when all channels are equally probable; conversely, it reaches its minimum when the distribution is concentrated on a single class, corresponding to a more confident prediction.
As anticipated in Sec. 2.4.5, in order to deal with imbalanced datasets, it is possible to adopt a more suitable loss function, such as the Focal Loss (F) [19], which reduces the relative weight of the most common labels:

$$\ell^{(F)}(Z;W) = -\sum_{c=1}^{C} Y_c^{(n)} \left(1 - \hat{Y}_c^{(n)}\right)^{\gamma} \log\left(\hat{Y}_c^{(n)}\right) \tag{3.16}$$

where $\gamma \geq 0$ is the focusing parameter that controls the strength of the modulation. When $\gamma = 0$, the focal loss reduces to the cross-entropy loss. This loss function is particularly used in object detection and anomaly detection tasks.
Accordingly, the empirical risk associated with cross-entropy over the dataset $\mathcal{T}$ is:

$$R_{\mathcal{T}}^{\mathrm{CE}}(W) = \frac{1}{N} \sum_{n=1}^{N} \ell^{(CE)}(Z;W) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} Y_c^{(n)} \ln\left(\hat{Y}_c^{(n)}\right) \tag{3.17}$$

This loss penalizes predictions that assign low probability to the correct label. In particular, if the predicted probability associated with the true positive label is close to 1, the loss approaches 0, conversely it becomes large. Therefore, cross-entropy encourages the network to produce predictions that are both accurate and confident.
By minimizing empirical risk evaluated with cross-entropy $R_{\mathcal{T}}^{\mathrm{CE}}(W)$ during training, the network learns discriminative features that improve class separation and overall recognition performance.

3.2.3 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence is conceptually related to to cross-entropy but is defined differently:

$$\ell^{(KL)}(Z;W) = \sum_{c=1}^{C} Y_c^{(n)} \ln\left(\frac{Y_c^{(n)}}{\hat{Y}_c^{(n)}}\right) \tag{3.18}$$

Then, the corresponding empirical risk is:

$$R_{\mathcal{T}}^{(KL)}(W) = \frac{1}{N} \sum_{n=1}^{N} \ell^{(KL)}(Z;W) = \frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} Y_c^{(n)} \ln\left(\frac{Y_c^{(n)}}{\hat{Y}_c^{(n)}}\right) \tag{3.19}$$

This quantity is always non-negative and is equal to zero if and only if the predicted distribution exactly matches the target distribution.
KL divergence is closely related to cross-entropy through the following identity:

$$\ell^{(CE)}(Z;W) = H\left(Y^{(n)}\right) + \ell^{(KL)}(Z;W) \tag{3.20}$$

where $H\left(Y^{(n)}\right)$ is the Shannon entropy of the target distribution:

$$H\left(Y^{(n)}\right) = -\sum_{c=1}^{C} Y_c^{(n)} \ln\left(Y_c^{(n)}\right) \tag{3.21}$$

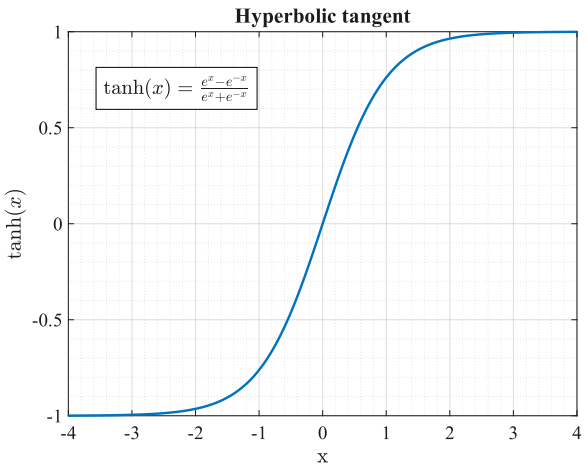
Since $H(Y^{(n)})$ depends only on the target distribution and not on the trainable parameters W, minimizing cross-entropy is equivalent to minimizing KL divergence up to an additive constant. In this sense, the optimization process drives the CNN to produce probability distributions that are increasingly consistent with the true modal class labels.

3.3 Activation Functions

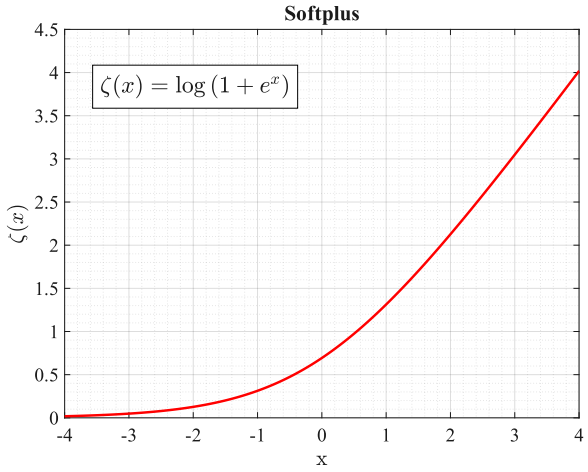
Activation functions represent the final transformation applied by individual neurons in a neural network. They provide a mathematical mechanism that enhances relevant features while attenuating less informative ones, thereby improving the expressive power and reliability of the network output.
In fact, neuron computations are essentially linear operations of the form:

$$Y = W \cdot X + b \tag{3.22}$$

Without nonlinear activation functions, the model would be unable to model complex nonlinear relationships between inputs and outputs. Hence, activation functions are essential to increase the representational capacity of the network. Moreover, different activation functions exhibit different behaviors: some saturate for large positive or negative inputs, while others preserve stronger gradients over wider input ranges. These properties directly affect training and, consequently, the overall learning capability of the model [23].
Among the activation functions commonly adopted in neural networks, the **hyperbolic tangent** and the **softplus** are widely used as internal nonlinear transformations. The hyperbolic tangent maps the input into the interval $(-1, 1)$ and is zero-centered, which can be advantageous during optimization, since its derivative assumes relatively large values in the neighborhood of $x = 0$ [26]. However, for very large positive or negative inputs, the derivative becomes small, possibly leading to saturation effects. Softplus, instead, provides a smooth and differentiable alternative, with a less abrupt transition around zero. It can be interpreted as a smooth version of ReLU and is useful when continuity and differentiability are desirable over the entire input domain.

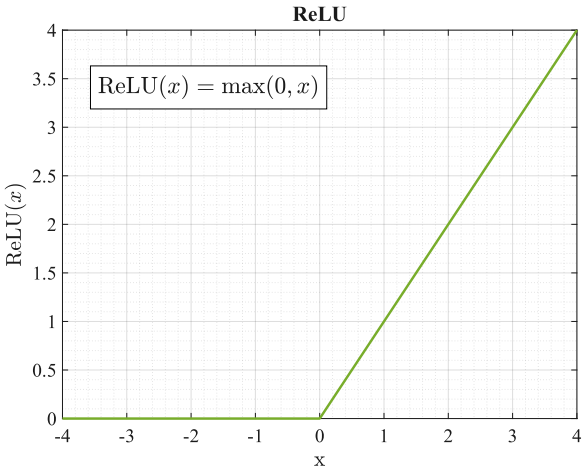


(a) Hyperbolic tangent activation function.

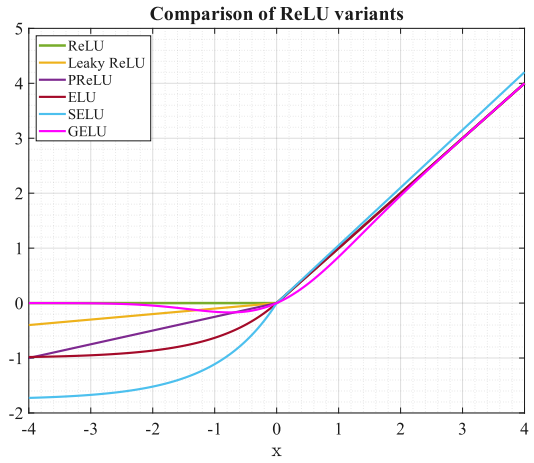


(b) Softplus activation function.

Another fundamental family is represented by the Rectified Linear Unit or **ReLU** function and its variants. ReLU is one of the most widely used activation functions in hidden layers because of its simplicity, computational efficiency, and favorable gradient behavior for positive inputs. It has been particularly successful in deep learning applications, especially in image recognition problems. However, several variants have been introduced to mitigate some of its limitations, especially for negative input values, where the standard ReLU output is exactly zero and the corresponding gradient vanishes. All the functions are designed to preserve a small nonzero response in the negative region. Softplus can be interpreted as a smooth approximation of the ReLU.

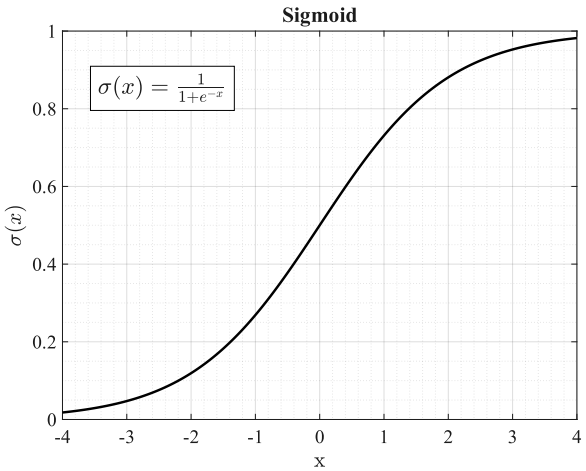


(a) ReLU activation function.

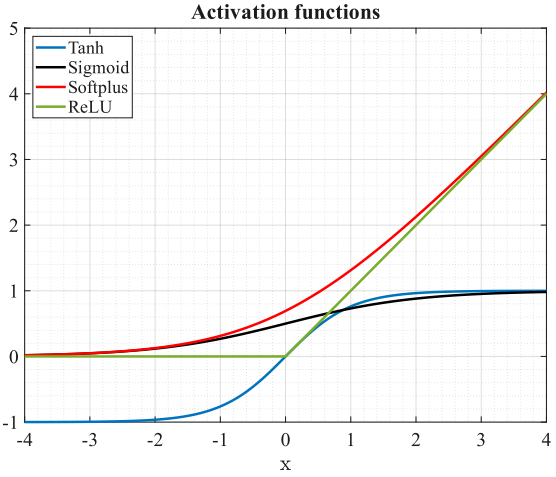


(b) ReLU family of functions.

Conversely, **sigmoid** can be used either as an internal activation function or as an output activation function. In binary and multi-label classification problems, it is particularly suitable at the output layer, since each output channel can be interpreted independently as the estimated probability that a specific label is present. This is especially important in multi-label settings, where multiple labels may coexist for the same input sample. Compared with tanh, sigmoid maps the input into the interval $(0, 1)$, which makes it naturally compatible with probabilistic interpretations. However, it is not zero-centered and may also suffer from saturation for large input magnitudes, resulting in small gradients. Nevertheless, its bounded output makes it a standard choice whenever the model is required to produce confidence values associated with the presence or absence of labels.

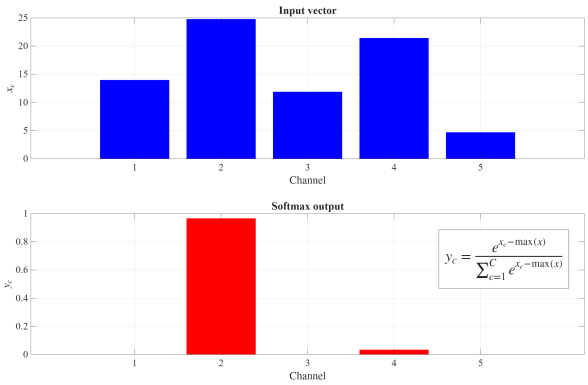


(a) Sigmoid activation function.

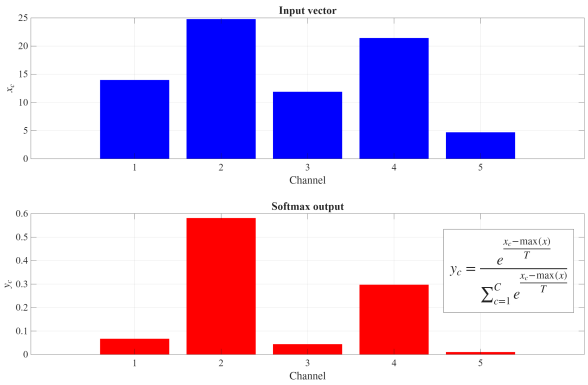


(b) Activation functions comparison.

Finally, **Softmax** is an output activation function that transforms the raw numerical scores produced by the network into a vector representing the probability associated with each class. In other words, softmax converts logits into a normalized probability distribution, making the output easier to interpret from both a physical and a decision-making perspective. This function is particularly suitable for multi-class classification problems, where the classes are mutually exclusive and exactly one class must be selected. Unlike sigmoid, which treats each output independently, softmax introduces a competition mechanism among the output channels, since increasing the probability of one class necessarily decreases the probability assigned to the others.
To mitigate the problem of excessive confidence or margin amplification, it is possible to introduce temperature scaling. This operation consists of normalizing the input logits by a positive scalar quantity before applying the softmax function. A higher temperature produces a softer probability distribution, reducing the dominance of the largest logit and assigning relatively more weight to lower-score classes.



(a) Physical meaning of the softmax function.



(b) Temperature ($T = 5$) effects on outputs.

3.3.1 Threshold

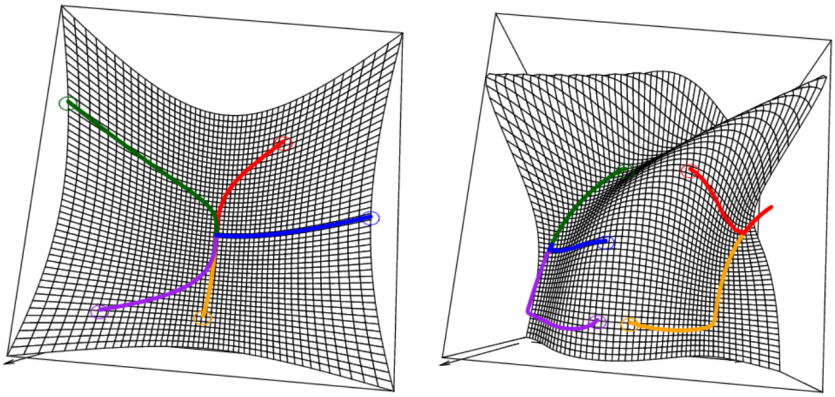
Once model outputs have been obtained, a threshold can be introduced to make the final decision: if the value associated with a given label exceeds the selected threshold, that label is assigned to the input; otherwise, it is rejected. Therefore, the threshold plays a crucial role in controlling the trade-off between false positives and false negatives, and its value can be selected empirically according to the requirements of the specific application. Different strategies can be adopted to determine the threshold, such as maximizing one of the evaluation metrics introduced in Sec.2.4.2.
In this thesis, the threshold has been computed by maximizing the $Micro - F_1$ metric, which is a common choice in multi-label classification when a single global threshold is adopted for all output channels. In fact, $Micro - F_1$ [24]:

- gives greater importance to the most frequent classes, since it aggregates the contributions of all labels before computing the final score;
- provides a measure of the overall performance of the system, rather than treating each class independently, that is particularly suitable when the objective is to determine a single operational threshold for the whole model;
- is generally more stable in the presence of class imbalance.

After thresholding, the dominant channels of the vector $\hat{Y}^{(n)}$ can be identified. These channels are then post-processed by a dedicated function to obtain the corresponding alphabetical string representation $\hat{y}^{(n)}$.

3.4 Gradient-Based Optimization Methods

The Empirical Risk Minimization (ERM) problem is generally not solvable analytically and can be formulated as an unconstrained optimization problem. Therefore, iterative optimization methods such as Gradient Descent (GD) and Stochastic Gradient Descent (SGD) are adopted in order to minimize $R_{\mathcal{T}}(W)$ with a computational cost compatible with practical applications. However, applications of Newton-based algorithms are not uncommon.
These methods are commonly introduced in the context of convex optimization. Despite the lack of global optimality guarantees in the non-convex case, gradient-based methods are still widely adopted, since they have proven effective in finding satisfactory solutions in high-dimensional optimization problems, as illustrated in Fig. 3.6.



(a) Convex function (b) Non-convex function

Figure 3.6: GD paths on convex (a) and non-convex (b) objective functions, obtained from different initial conditions [14].

Before introducing the different variants, it is useful to clarify the terminology commonly used in machine learning optimization:

- an **iteration** is a single weight update;
- the **batch size** is the number of training samples used to estimate the gradient in one iteration;
- an **epoch** is one complete pass over the entire training set.

Therefore, in Batch GD the batch size coincides with the number of samples in the entire training set, and one iteration corresponds to one epoch. In SGD, the batch size is equal to one, and many iterations are required to complete one epoch. In Mini-batch SGD, the batch size is intermediate between these two cases, and each epoch consists of multiple iterations.

3.4.1 Batch Gradient Descent

Batch Gradient Descent (Batch GD) is one of the simplest and most intuitive first-order methods for solving minimization problems. Once the ERM problem and the loss model have been defined, the solution $W^\star \in \mathcal{W}$ can be identified as the minimizer of $R_{\mathcal{T}}(W)$, which is denoted by g in the following discussion.

Based on the descent direction defined by the negative gradient, the sequence [14] is:

$$W^{(k)} = W^{(k-1)} - lr^{(k)} \nabla g\left(W^{(k-1)}\right) \tag{3.23}$$

where $k \in \mathbb{N}$ denotes the iteration index, while $lr^{(k)}$ is the step size at iteration k, called the **learning rate** in the deep learning literature. The initial guess $W^{(0)}$ is typically chosen according to initialization criteria that may influence the convergence behavior. Consider the first-order approximation:

$$g\left(W^{(k)}\right) = g\left(W^{(k-1)} - lr^{(k)} \nabla g\left(W^{(k-1)}\right)\right) \approx g\left(W^{(k-1)}\right) - lr^{(k)} \left\|\nabla g\left(W^{(k-1)}\right)\right\|_2^2 \tag{3.24}$$

Let us define the squared L_2-norm as:

$$L_2^2 \doteq \left\|\nabla g\left(W^{(k-1)}\right)\right\|_2^2 = \nabla g\left(W^{(k-1)}\right)^\top \nabla g\left(W^{(k-1)}\right) \geq 0 \tag{3.25}$$

Therefore, for a sufficiently small learning rate, the sequence is monotonically decreasing:

$$g\left(W^{(k)}\right) < g\left(W^{(k-1)}\right) \tag{3.26}$$

Hence, the learning rate is one of the most important **hyperparameters**, i.e. parameters that govern the training phase: its choice influences, in particular, the speed at which the solver approaches a solution and also the presence of oscillations in the loss path, which should be avoided. This latter phenomenon is related to the stability of the method and may occur when the step size is too large, thus preventing convergence. In general, the choice of hyperparameters is often performed through trial and error, as shown in Fig. 3.7.

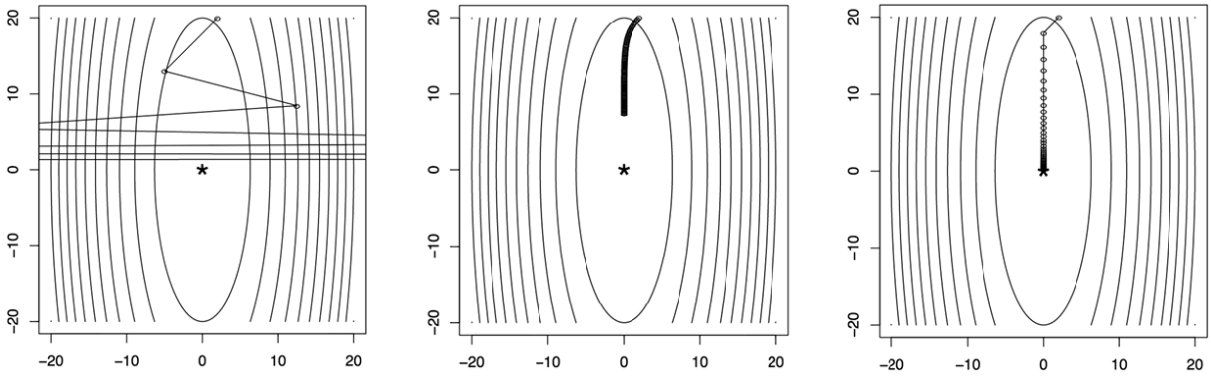


Figure 3.7: Illustration of learning-rate empirical loop[14].

In iterative algorithms, the solution is reached only when a prescribed stopping criterion is satisfied; this may consist of limiting the maximum number of iterations or requiring the relative error to stay below a given tolerance:

$$\varepsilon \doteq \frac{\|W^{(k)} - W^*\|_2}{\|W^{(0)} - W^*\|_2} \tag{3.27}$$

In general, Batch GD is easy to implement and performs well for strictly convex problems, although its convergence speed is typically rather slow. In fact, for convex differentiable functions with Lipschitz-continuous gradient, the number of iterations required to reach an accuracy ε is $\mathcal{O}(1/\varepsilon)$ [14].

Nevertheless, the empirical risk encountered in practical applications is usually non-convex. Moreover, when the objective function is characterized by a large condition number, the corresponding optimization problem is said to be ill-conditioned. The condition number quantifies the sensitivity of the output to perturbations in the input. In such cases, the level sets of the objective function are highly elongated, and the convergence of Batch GD may become slow and oscillatory.
A further disadvantage is that in Batch GD the gradient of the empirical risk is computed using all the N training samples at each iteration [17]. This computation can become very expensive in deep learning applications, where N is typically very large. As a consequence, although Batch GD provides the basic conceptual framework for gradient-based learning, its direct application to large-scale problems is often impractical.

3.4.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is one of the state-of-the-art optimization methods in machine learning and deep learning. The iterative sequence is formally analogous to Eq. 3.23; however, its main advantage lies in replacing the exact gradient of the functional g with a stochastic estimator [17], whose expectation coincides with the exact gradient:

$$\mathbb{E}\left[\tilde{\nabla} g\left(W^{(k-1)}\right)\right] = \nabla g\left(W^{(k-1)}\right) \tag{3.28}$$

Therefore:

$$W^{(k)} = W^{(k-1)} - lr^{(k)} \tilde{\nabla} g\left(W^{(k-1)}\right) \tag{3.29}$$

In its pure form, SGD uses a batch size equal to one, i.e. each update is computed using only one randomly selected training sample, resulting in a very low computational cost per iteration. This makes SGD particularly attractive in large-scale learning problems.
However, the resulting gradient estimate is noisy, and the optimization trajectory is typically more irregular. As a consequence, SGD often reaches a neighborhood of a minimum quickly, but may have difficulty converging to it. Overall, GD exhibits a slow convergence rate, whereas SGD is computationally efficient and accurate but lacks of precision.

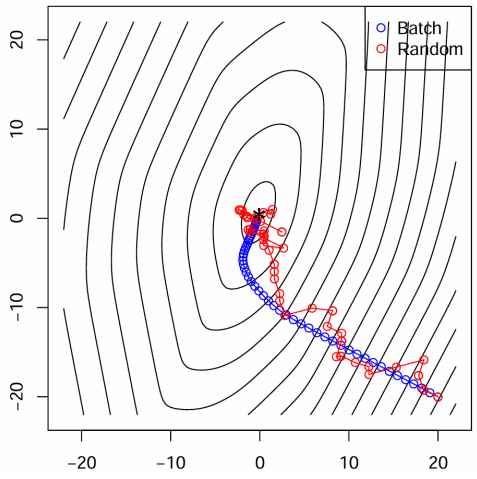


Figure 3.8: Comparison between Batch GD and SGD[27].

3.4.3 Mini-batch Stochastic Gradient Descent

Mini-batch Stochastic Gradient Descent represents an intermediate approach between Batch GD and pure SGD, and it is the method most commonly adopted in practical deep learning applications.
Instead of using either the full training set or a single sample, Mini-batch SGD considers a stochastic subset $\mathcal{B} \subset \mathcal{T}$, characterized by batch size $1 < B < N$. The optimization is then performed using the empirical risk restricted to the mini-batch, namely:

$$g_{\mathcal{B}}(W) \doteq R_{\mathcal{B}}(W) \tag{3.30}$$

The corresponding sequence can be written as:

$$W^{(k)} = W^{(k-1)} - lr^{(k)} \tilde{\nabla} g_{\mathcal{B}} \left(W^{(k-1)} \right) \tag{3.31}$$

To summarize, Mini-batch SGD offers a compromise between Batch GD and pure SGD, because reduces the computational cost of each update and provides a less noisy gradient estimate.
However, there are many preferable variants. The most well-known is ***Adam***, a computationally efficient optimization algorithm that is implemented in many commercial optimization software packages and deep learning libraries. Qualitatively, Adam can be interpreted as an adaptive variant of stochastic gradient-based methods. In fact, the updates are smoothed by adding an artificial momentum to the gradient [28]:

$$\begin{aligned} W^{(k)} &= W^{(k-1)} - lr^{(k)} \gamma^{(k+1)} \\ \gamma^{(k+1)} &= \mu \gamma^{(k)} + (1-\mu) \tilde{\nabla} g \left(W^{(k-1)} \right) \end{aligned} \tag{3.32}$$

where γ is the modified gradient and $\mu \in \mathbb{R}(0,1)$ is the momentum factor.

3.5 Newton-Based Optimization Methods

In general, Newton-based algorithms are characterized by a high convergence rate [25]. Unlike first-order gradient-based methods, they exploit second-order information. In fact, by considering the second-order Taylor expansion of the ER with increment ΔW:

$$g(W + \Delta W) \approx g(W) + \Delta W^T \nabla g(W) + \frac{1}{2} \Delta W^T H(W) \Delta W \tag{3.33}$$

where $H(\cdot)$ denotes the Hessian matrix:

$$H(W) = \nabla^2 g(W) = \frac{\partial^2 g(W)}{\partial W \partial W^T} \tag{3.34}$$

Minimizing the quadratic approximation in Eq. 3.33 with respect to ΔW leads to:

$$\Delta W^{(k)} = -H^{-1} \left(W^{(k-1)} \right) \nabla g \left(W^{(k-1)} \right) \tag{3.35}$$

and therefore to the Newton update:

$$W^{(k)} = W^{(k-1)} - H^{-1} \left(W^{(k-1)} \right) \nabla g \left(W^{(k-1)} \right) \tag{3.36}$$

From a geometric point of view, Eq. 3.36 can be interpreted as the minimization of a local second-order approximation of the objective function around the current iterate. In other words, at each iteration, the function is locally approximated by a quadratic model, and the variables are updated by moving toward the minimum of this approximation.

3.6 Weight Initialization Algorithms

Weight initialization is a crucial step in training deep neural networks, since it affects the stability and convergence of the optimization process. A poor initialization may lead to oscillations, slow convergence, or unstable training.
In general, initialization methods can be divided into a few main classes:

- Zero initialization, where weights are set to zero;
- Random initialization, which consists in assigning values drawn from a probability distribution;
- Variance-preserving initialization, where the variance of the initialized weights is chosen so as to remain approximately constant across layers.

In general, zero initialization is mainly used for bias terms, since their optimized values are typically small. For weights, however, zero initialization is avoided because it does not break symmetry among neurons. Among variance-preserving methods, Xavier-Glorot initialization is commonly used with saturating activation functions such as *tanh*, while He initialization is better suited for ReLU-based activations.
The **Xavier-Glorot** initialization [30] is designed to keep the variance of activations approximately constant across layers. Neural network are composed by layers, each layer l receives a set of n_{in} input units and produces a set of n_{out} output units. Hence, corresponding weight $W^{(l)}$ can be initialized according to the Xavier normal scheme as:

$$W^{[l]} \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right) \tag{3.37}$$

where $\sim \mathcal{N}(0, \sigma^2)$ means that values are randomly taken ($\sim$) from a Gaussian normal ($\mathcal{N}$) distribution with mean 0 and variance σ^2.
This initialization is typically recommended for activation functions such as *tanh* or other sigmoid-like nonlinearities.
There is a corresponding uniform ($\mathcal{U}$) [32] variant, where each weight is initialized with values from uniform distribution rather than according to a Gaussian distribution:

$$W^{[l]} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, +\sqrt{\frac{6}{n_{in} + n_{out}}}\right) \tag{3.38}$$

Instead, the **He** [29] initialization is specifically designed for ReLU activations and their variants, since these functions tends to azzerate negative inputs that can lead to some optimisation problems. To compensate for this effect, a larger variance is used:

$$W^{[l]} \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right) \tag{3.39}$$

While, in the uniform version:

$$W^{[l]} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}}\right) \tag{3.40}$$

This initialization is commonly used with ReLU, Leaky ReLU, and related activation functions.
Integer values can be adjusted depending on the activation function by scaling with gains[31].

4 - AI models

4.1 Pipeline Overview

The development of an AI model can be divided into four main phases:

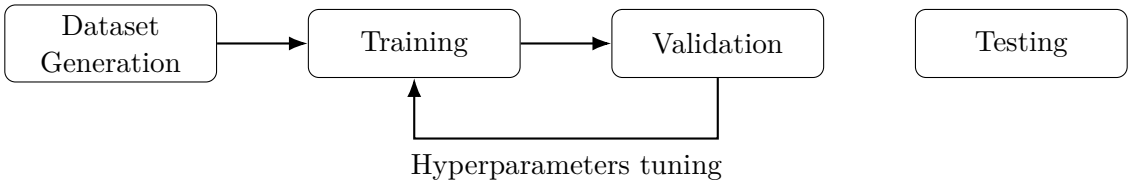


Figure 4.1: Block diagram of the AI model development pipeline.

Dataset processing is one of the most critical stages of the pipeline, since both the input data and the corresponding labels must be handled carefully. In the present project, particular attention must also be devoted to the oversampling procedure, which must be performed cautiously in order to avoid introducing physically unrealistic alterations. Moreover, the labelling process is more challenging than in many similar classification tasks.
Another source of complexity arises from the nature of the input X. In the blade modal shape classification problem, the input is a complex-valued three-dimensional tensor representing modal displacement, defined as:

$$X \in \mathbb{C}^{n \times n \times 3} \tag{4.1}$$

Once the input matrix X and the target matrix Y have been constructed, the next phase is training, during which the model learns to associate each input with its corresponding output, producing a prediction vector $\hat{Y}$. In this context, hyperparameters are the configuration variables that define both the model architecture and the training procedure.
Before the final evaluation, a validation phase is introduced in order to tune the hyperparameters and improve the model performance through an iterative training–validation loop.
Finally, the testing phase is carried out to assess the performance of the trained model and compute the relevant metrics on different case studies.

4.1.1 Training and Inference Time

Time is one of the main parameters governing optimization analyses in AI, and in this context two distinct definitions must be considered.
Training time is the computational time required by the AI framework to learn from the available data and optimize its internal parameters. Inference time, by contrast, is the time required by the trained model to produce an output for a new input sample.

Algorithm	Training	Classification
Naïve Bayes	$\mathcal{O}(nF)$	$\mathcal{O}(F)$
Decision Tree	$\mathcal{O}(n^2F)$	$\mathcal{O}(F)$
SVM (kernel based)	$\mathcal{O}(n^2F + n^3)$	$\mathcal{O}(n_{SV}F)$
k-NN	–	$\mathcal{O}(nF)$
Linear Regression	$\mathcal{O}(F^2n + F^3)$	$\mathcal{O}(F)$
Random Forest	$\mathcal{O}(n^2F\, n_{\text{trees}})$	$\mathcal{O}(F\, n_{\text{trees}})$

Symbol	Meaning
n	data points
F	features
n_{trees}	trees
n_{SV}	support vectors

Figure 4.2: Computational complexity of the principal ML algorithms [6].

4.2 ML models

The definition of Machine Learning also includes Deep Learning, since both ultimately aim to identify patterns within sets of features. More generally, Machine Learning includes predictive models that can be implemented on a computer in order to infer outputs from input feature data.
Therefore, in this section, the inputs are intended as feature spaces that have already been extracted manually or through dedicated preprocessing code. As consequence, assuming that each label is associated with sufficiently distinctive features, the samples can be represented in the feature space as groups occupying different regions, separated according to their feature values.
The simplest example of a machine learning model is linear regression [1], which also represents the basic computation performed by an artificial neuron:

$$\hat{Y} = \hat{f}(X; W) = W^T X + b \tag{4.2}$$

Machine learning algorithms are highly diverse in terms of assumptions, structure, and optimization strategy. A fundamental result in learning theory is the **No Free Lunch** (NFL) theorem [33], according to which no learning algorithm is universally superior when performance is averaged over all possible problems. This implies that the effectiveness of a model depends on the characteristics of the input data and on the nature of the output to be predicted.
However, ML models can generally be divided into two broad groups:

- **Parametric models**, in which the number of parameters is finite and fixed once the training process has been completed;
- **Non-parametric models**, which are mainly based on the relative position of data points, often relying on neighborhood relationships, and classify new samples by evaluating distances in the feature space.

In general, identifying the most representative features among the potentially many parameters describing a problem is one of the most challenging tasks in machine learning. Nevertheless, suitable transformations of the input space may map samples into representations in which the classes become more easily separable. Rather than relying exclusively on manually designed transformations, representation learning aims to automatically discover such informative representations directly from data [1].

4.2.1 Decision Trees

In the features space, a decision tree partitions the feature space by means of a sequence of simple decision rules. Each partition corresponds to a branch of the tree [33], while each internal node is associated with a condition, typically expressed as an inequality on one of the input features. At each node, the algorithm selects a feature and a threshold in order to split the dataset into two subsets. By recursively repeating this procedure, the feature space is divided into increasingly homogeneous regions. The terminal nodes, or leaves, contain the output label assignment.
One of the main advantages of decision trees is their interpretability, since the classification process can be described through a set of explicit logical rules that are easy for humans to read and understand.

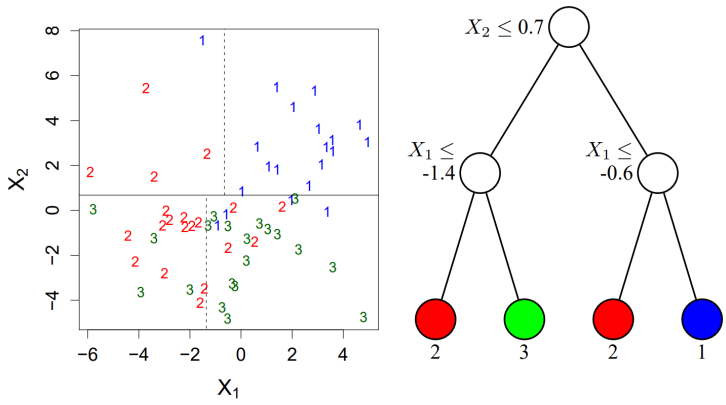


Figure 4.3: Decision three[33].

4.2.2 Support Vector Machines

Support Vector Machines (SVMs) are a family of classification methods based on decision boundaries called hyperplanes [35]. In a high-dimensional input space, these hyperplanes are constructed in order to separate samples belonging to different classes.
The key concept underlying SVMs is the margin, i.e., the maximum distance between two hyperplanes parallel to the separating hyperplane that do not intersect any training sample. The optimal hyperplane is therefore placed midway between the closest groups of samples, so as to maximize the separation margin. The samples lying closest to the margin boundaries are called support vectors, since they determine the position of the separating hyperplane.

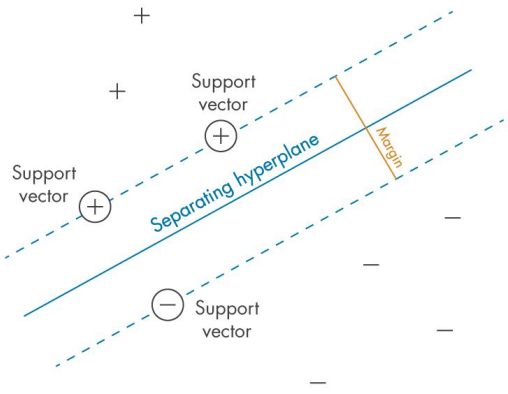


Figure 4.4: Hyperplane and margin representation [35].

SVMs are characterised by:

- the ability to operate effectively in high-dimensional feature spaces;
- good robustness, especially when the number of features is large;
- widespread application in many fields, particularly in natural language processing and image classification;
- the possibility of being implemented in either linear or nonlinear form, although the computational cost increases with the degree of nonlinearity and with the number of features;
- performance that may deteriorate in the presence of noisy data.

One of the main advantages of SVMs is the possibility of using the kernel trick, namely a mathematical transformation that maps the input data into a higher-dimensional feature space in which class separation becomes easier.

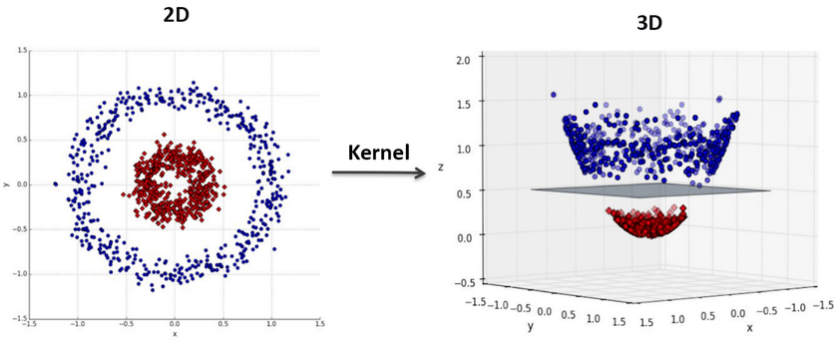
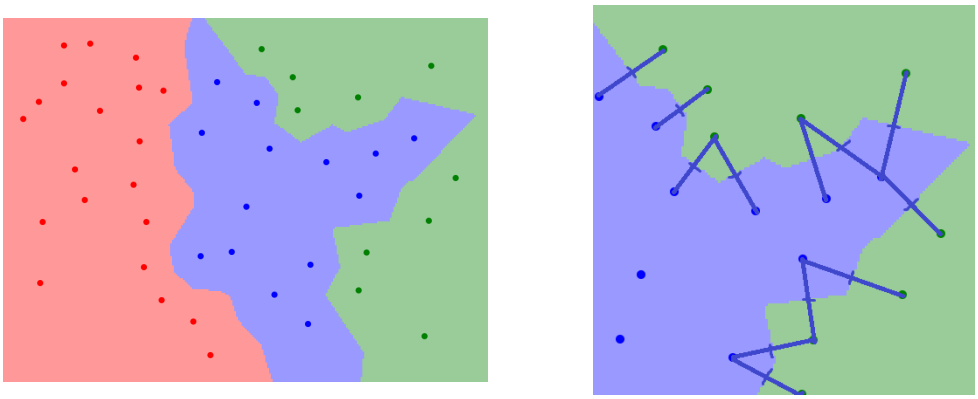


Figure 4.5: Kernel trick[36].

4.2.3 k-Nearest Neighbor Classifier

k-Nearest Neighbor (k-NN) classifiers are distance-based methods that assign a class label to a new sample according to the labels of its k closest training samples in the feature space [37]. More precisely, once a distance metric has been selected, the algorithm identifies the neighborhood of the query point and performs the classification on the basis of similarity.



(a) Query points.

(b) Neighbor construction.

Figure 4.6: k-NN [38].

If $k = 1$, the predicted label coincides or not with that of the single nearest training sample. For $k > 1$, instead, the prediction is typically obtained by majority voting among the k nearest neighbors. Among the possible definitions, the Minkowski distance provides a general formulation that includes several well-known metrics as special cases. Given two samples $X_A, X_B \in \mathbb{R}^F$, where F is the number of features, the Minkowski[39] distance is:

$$d_{AB} = \left(\sum_{f=1}^{F} \left| X_A^{(f)} - X_B^{(f)} \right|^p \right)^{1/p} \tag{4.3}$$

where $p \geq 1$ is a real parameter controlling the type of distance. Different choices of p lead to different definitions, as summarized in Table4.1. Some k-NN variants consider also a weight associated with each neighbor[40].

Distance metric	**Value of p**	**Formula**
City block	$p = 1$	$d_{AB} = \sum_{f=1}^{F} \left\| X_A^{(f)} - X_B^{(f)} \right\|$
Euclidean	$p = 2$	$d_{AB} = \left(\sum_{f=1}^{F} \left\| X_A^{(f)} - X_B^{(f)} \right\|^2 \right)^{1/2}$
Chebyshev	$p \to \infty$	$d_{AB} = \max_f \left\| X_A^{(f)} - X_B^{(f)} \right\|$

Table 4.1: Special cases of the Minkowski distance [39].

Another widely used similarity-based measure is the cosine distance, which is particularly useful when the orientation of the feature vectors is more relevant than their magnitude:

$$d_{AB} = 1 - \frac{X_A \cdot X_B^T}{\sqrt{(X_A \cdot X_A^T)\,(X_B \cdot X_B^T)}} \tag{4.4}$$

4.2.4 Ensemble Classifiers

Ensemble classifiers are hybrid methods that combine multiple weak learners [41], appropriately weighted or aggregated, in order to obtain a more accurate and robust final prediction. The underlying idea is that, although each individual classifier may be only moderately effective, their combined decision can significantly improve the overall classification performance.

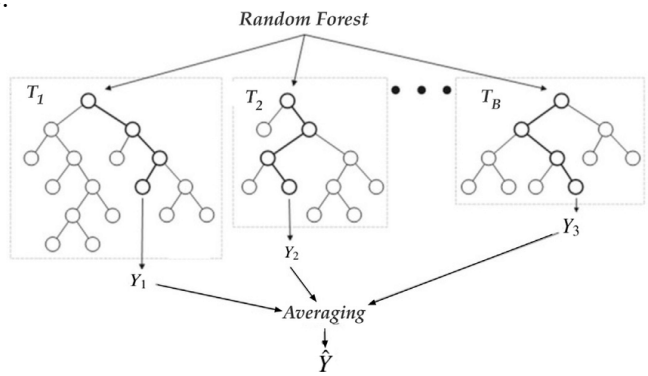


Figure 4.7: Random forest is an ensemble of decision trees[1].

[1]Random forest structure figure by K. Grolinger, available at: https://www.researchgate.net/profile/Katarina-Grolinger/publication/314651819/figure/fig2/AS:669026955042818@1536519864287/Random-forest-structure-16.ppm.

4.3 DL models

Deep Learning models are essentially on **artificial neural networks**, i.e., computational architectures inspired by biological neural systems. A neural network is composed of interconnected computational units, called neurons, which receive input signals, process them, and transmit the resulting outputs to subsequent units. The term deep itself refers to the presence of multiple layers, which enable the network to learn hierarchical feature representations, from simple low-level patterns to increasingly abstract and task-relevant features. By arranging neurons in sequence, a basic model is structured into **input layer**, one or more **hidden layers**, and an **output layer**.

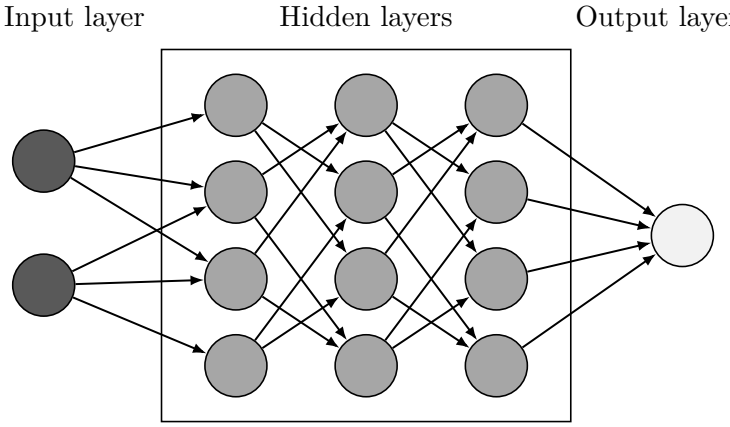


Figure 4.8: Scheme of a fully connected neural network with multiple hidden layers.

Hence, as anticipated in Eq. 3.22, the output of the i-th layer can be described according to the classical McCulloch-Pitts (MCP) neuron model [44] as:

$$H^{(i)} = \alpha^{(i)} \left(W^{(i)T} H^{(i-1)} + b^{(i)} \right) \tag{4.5}$$

where $W^{(i)}$ and $b^{(i)}$ are the weights and biases of the i-th layer, respectively, and $\alpha^{(i)}$ denotes the selected activation function. For $i = 1$, $H^{(0)} \equiv X$. This expression shows that each layer performs a linear transformation followed by a nonlinear operation, allowing the network to learn increasingly complex input-output relationships.
To illustrate how a neural network processes inputs such as vectors, matrices, or images, the example in Fig. 4.9 can be considered [1]:

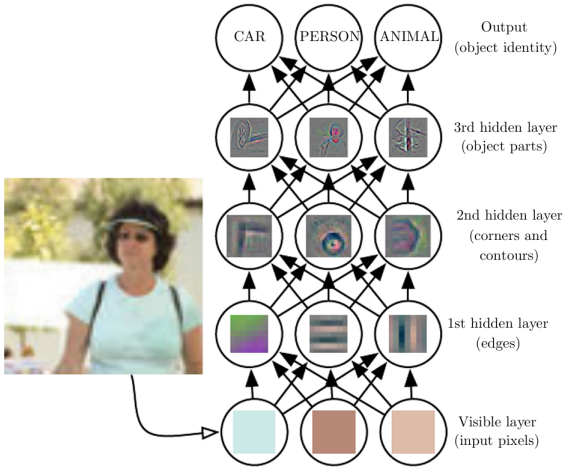


Figure 4.9: Illustration of how a neural network processes information.

Starting from the values provided at the input layer, the hidden layers progressively extract increasingly informative features by applying successive transformations. In image-based problems, for instance, early layers may detect simple patterns such as edges or local contrasts, whereas deeper layers can combine these elementary features into more abstract and task-relevant representations.
This architecture is based on **forward propagation**, namely the sequential transfer of information from the input layer to the output layer during training. Conversely, the parameters of the network are updated through the **backpropagation** algorithm, which computes the gradient of the loss function with respect to the weights, thereby allowing the error information to propagate backward through the network. The number of hidden layers determines the **depth** of the model: increasing the depth generally enhances the representational power of the network, although it also increases the computational cost and the complexity of the training process.
Depending on the structure of the input data and on the target task, different neural network architectures can be adopted. In the present work, the investigated models include some of the most widely adopted architectures in pattern and image recognition, namely Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Graph Neural Networks (GNNs).

4.3.1 GNN

Graph neural networks (GNNs) are algorithms designed to operate on graph-structured data [42]. In general, a graph is a structure that describes relationships between entities, called nodes, through connections called edges. Both nodes and edges can store information. This concept can be intuitively understood from the following example, in which an image is decomposed into interacting nodes associated with different local features:

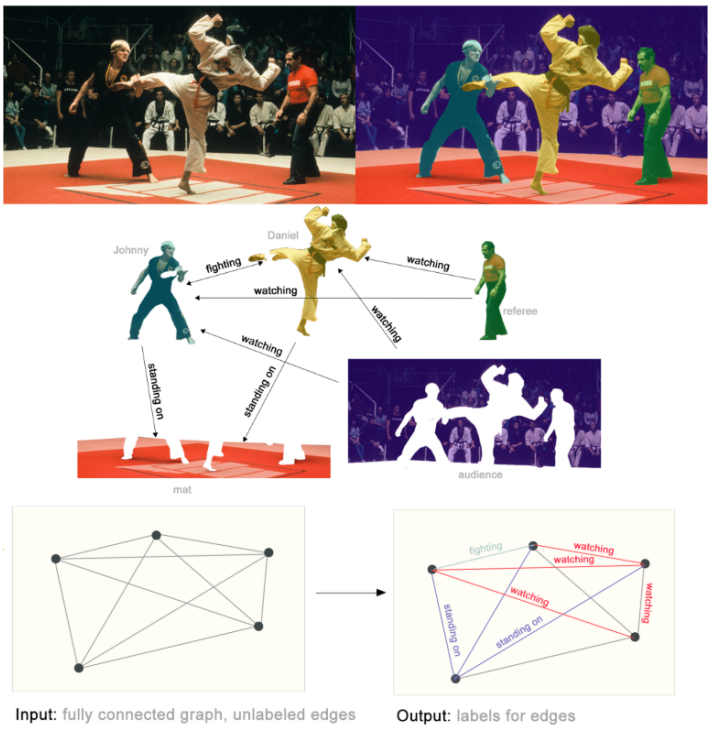


Figure 4.10: Example of graph construction based on image decomposition [43].

GNNs are widely adopted in geometric deep learning for pattern recognition and in chemistry for molecular classification, where atoms can be interpreted as nodes and chemical bonds as edges. Graphs can be represented through an adjacency matrix, whose entries are equal to 1 or 0 depending on whether two nodes are connected. For the purposes of this thesis, the graph is constructed directly from the FEM nodal position, providing a representation of the displacement distribution and its local spatial relationships, which are essential for the correct classification of modal shapes. This means that, independently of the specific CAD geometry of the blade, the modal displacement field can be mapped into a graph-based representation and then associated with a label, as shown below through a simple distance-thresholding algorithm:

$$A_{ij} = \begin{cases} 1, & \text{if } d_{ij} \leq r \\ 0, & \text{if } d_{ij} > r \end{cases} \qquad r \in \mathbb{R}^+$$

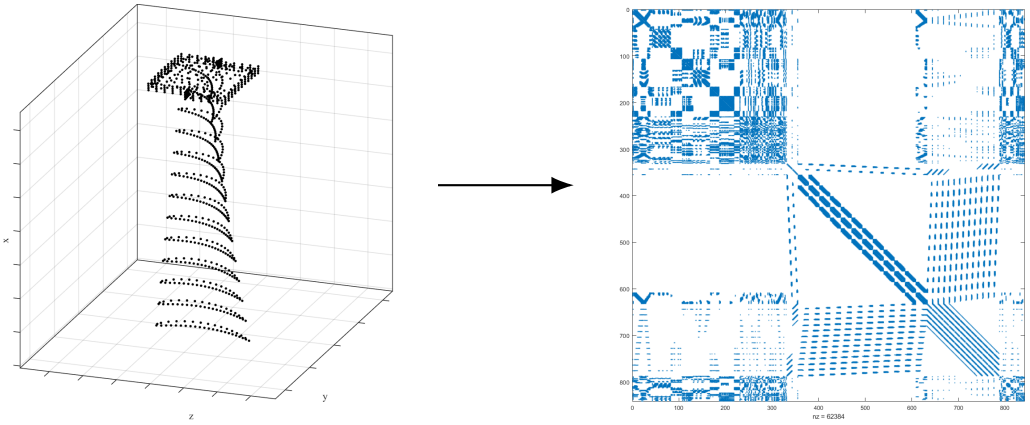


Figure 4.11: Construction of the adjacency graph according distance-threshold criterion.

Graph Convolutional Network (GCN) belongs to GNNs and consists of three main steps. First, the node features are linearly transformed through a trainable weight matrix:

$$\tilde{H}^{(i)} = H^{(i-1)}W^{(i)} + b^{(i)} \tag{4.6}$$

Then, the features are propagated across the graph through the adjacency matrix:

$$H^{(i)} = A\tilde{H}^{(i)} \tag{4.7}$$

Finally, a nonlinear activation function is applied.
Therefore, each node updates its representation by combining its own features with those of its neighboring nodes, according to the graph connectivity encoded in the adjacency matrix.
The main advantage of GNNs lies in their ability to process data defined on a non-Euclidean domain, where informations are not organised in Cartesian grid which cannot be directly handled by standard neural architectures, such as blades geometry. For that reason, they are implemented in different physics-based fields. Nevertheless, graph-based operations may become computationally expensive, since the adjacency matrix must be combined with node features and trainable weights in order to propagate information across the graph.

4.3.2 MLP

The McCulloch–Pitts (MCP) neuron, introduced in 1943, represents one of the earliest mathematical models of an artificial neuron. A subsequent development of this idea led to the perceptron, which can be interpreted as a single-layer architecture composed of multiple neurons and followed by an activation step.

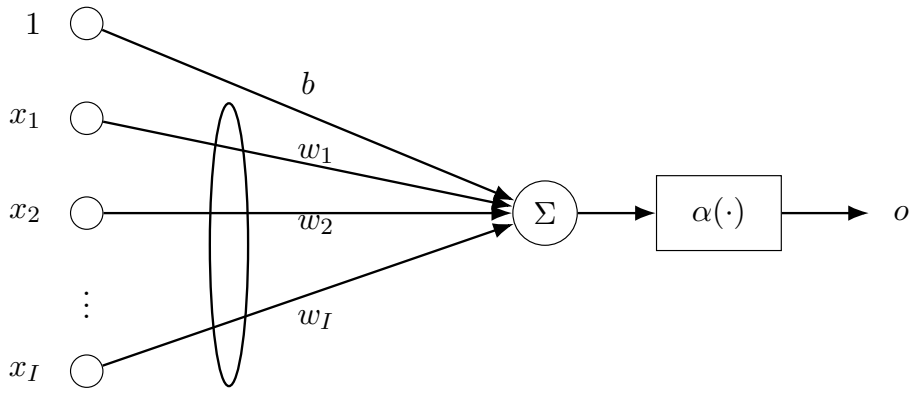


Figure 4.12: Single MCP neuron.

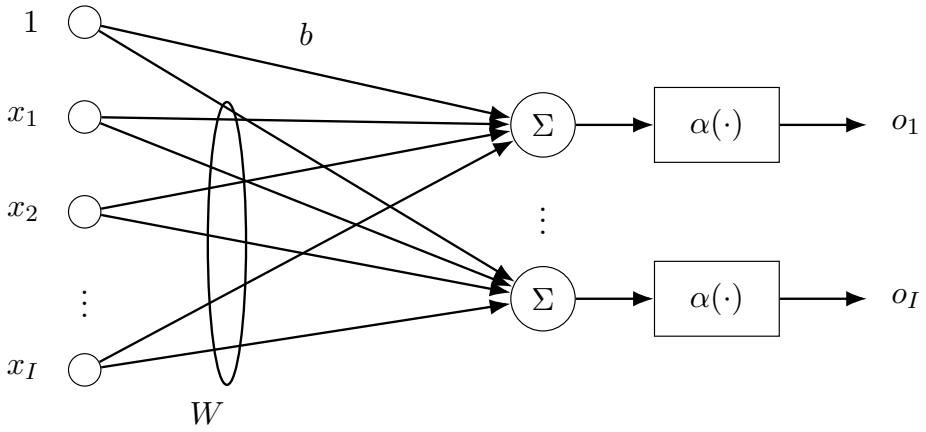


Figure 4.13: Single-layer perceptron classifier (SPC).

By stacking multiple perceptron layers, it is possible to obtain the family of **Multi-Layer Perceptrons (MLPs)**. These architectures are fully connected feed-forward neural networks in which each neuron of one layer is connected to all neurons of the subsequent layer. Owing to the presence of nonlinear activation functions, MLPs are able to approximate complex input-output relationships and therefore represent the standard extension of the single-layer perceptron to more challenging classification and regression tasks.

Compared with CNNs and GNNs, MLPs are generally simpler and computationally less expensive. However, they do not explicitly exploit the spatial structure of grid-based data or the relational structure of graph-based data, since they operate directly on individual inputs, and for this reason they are often less effective when such information plays a central role.

4.3.3 CNN

Convolutional Neural Networks (CNNs) represent the state of the art in image classification algorithms. Their structure is conceptually similar to that of MLPs, but enriched by the introduction of the **convolution** operation [1], usually denoted by $*$. This operation allows the network to extract local patterns from an input matrix or vector by means of a sliding-window mechanism. To achieve this, CNNs employ small trainable matrices, called *kernels* or *filters*, $K \in \mathbb{R}^{N \times M}$, which are applied to the generic layer input $H^{(i)}$ in order to produce a transformed output matrix $\tilde{H}^{(i)}$, as shown in Fig. 4.14.

$$\tilde{H}^{(i)} = H^{(i)} * K = \sum_{m=1}^{M} \sum_{n=1}^{N} H^{(i)}(j-m, k-n)\, K(m,n) \tag{4.8}$$

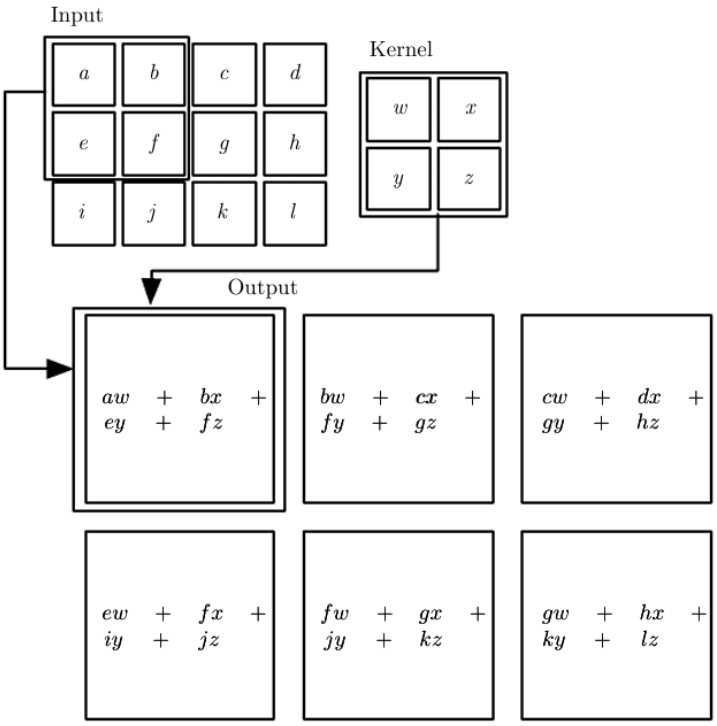


Figure 4.14: Qualitative convolution operation on a 2D input matrix [1].

There are some variants of this operation, depending, for instance, on whether the kernel is flipped or whether a cross-correlation is used instead of a strict convolution. In practical deep learning implementations, cross-correlation is often adopted, although the term convolution is conventionally retained.
By stacking multiple convolutional layers, CNNs are able to learn hierarchical representations, in which shallow layers extract simple local features, such as edges or intensity transitions, whereas deeper layers progressively combine them into more abstract and task-oriented patterns. Moreover, each convolution kernel generates a different feature map, so that multiple filters can capture complementary aspects of the same input.
These feature maps are often referred to as **channels**, since each of them stores a different learned representation of the input. In deeper layers, such channels can be interpreted as **latent channels**, because they no longer correspond to directly observable quantities, but rather to internal features automatically extracted by the network. Their role is to encode increasingly abstract and task-relevant information that can be exploited by the subsequent layers for classification.

At the end, a pooling operation is typically introduced in order to summarize the convolution results and reduce their dimensionality. In this phase, maximum or average values are extracted over local regions, producing more compact feature maps and improving robustness to small spatial variations before the final prediction stage.

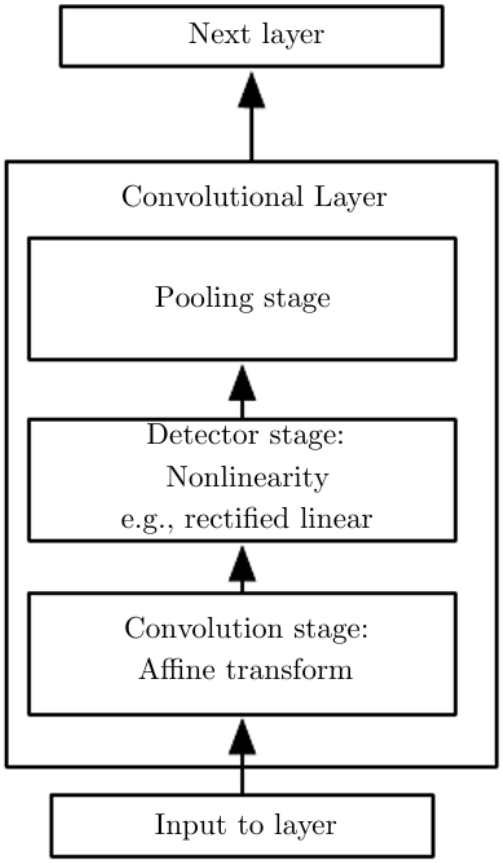
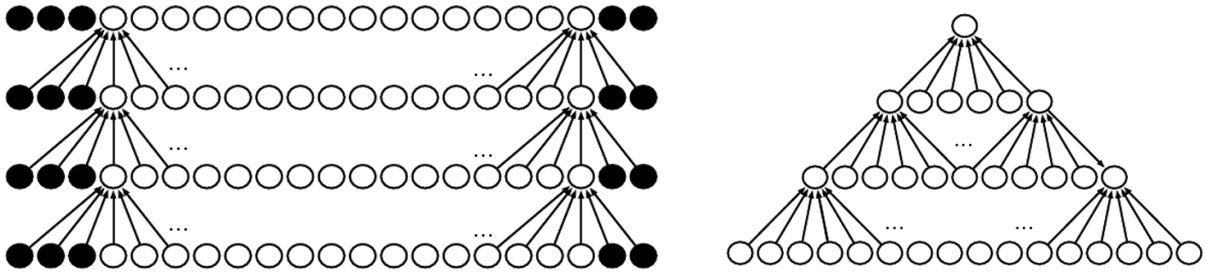


Figure 4.15: Anatomy of a convolutional neural network layer [1].

The behavior of a convolutional layer is also strongly influenced by two important design parameters, namely **stride** and **padding**. The stride defines the step with which the kernel slides across the input domain. A unit stride means that the filter is shifted by one element at a time, thus producing a dense output feature map and preserving as much spatial information as possible. By contrast, a larger stride reduces the spatial resolution of the output, since some values are considered less than others. Therefore, increasing the stride decreases the computational cost and the output size, but it may also lead to a partial loss of local detail.
Padding, on the other hand, consists in extending the input matrix by adding extra values, usually zeros, around its boundary before the convolution is performed. This operation is particularly important because, without (zero) padding, the spatial dimensions of the feature maps tend to decrease after each convolutional layer. As a consequence, repeated convolutions may quickly shrink the representation and discard useful information near the borders of the input.



(a) Convolution with padding. (b) Convolution without padding.

Figure 4.16: Comparison of the effect of padding on the spatial dimensions of the output feature map[1].

More precisely, if the input has spatial size L, the kernel has size F, the stride is S, and the padding size is P, the output size can be expressed as:

$$L_{\text{out}} = \left\lfloor \frac{L - F + 2P}{S} \right\rfloor + 1 \tag{4.9}$$

This relation shows that stride and padding directly determine the dimensions of the output feature map. In particular, using larger padding values helps preserve the original spatial extent, whereas increasing the stride produces a more aggressive downsampling effect. For this reason, the choice of stride and padding is not merely technical, but has a direct impact on the trade-off between computational efficiency and retention of spatial information.
In practical CNN design, these parameters are selected according to the nature of the task. When fine local detail is important, small strides and suitable padding are generally preferred, whereas larger strides may be useful when dimensionality reduction is required already in the early layers.
Convolution leverages several important principles that improve the effectiveness of machine learning systems:

- Sparse interactions, since each output value depends only on a local portion of the input. As a consequence, the network can assign different importance to specific local patterns, such as edges, gradients, or texture variations, rather than treating all input values independently;
- Parameter sharing, since the same kernel, typically small in size, is applied across the entire input domain;
- Equivariant representations, since a spatial shift in the input produces a corresponding shift in the output feature map;
- Robustness, since convolution and pooling combined provide a natural way to process inputs with structured spatial organization and can be adapted to different input sizes.

Those properties make CNNs particularly effective in image analysis, where local spatial structures carry most of the relevant information.
Finally, the computational cost of the convolutional operation is proportional to the kernel size and to the size of the processed inputs.

5 - Bladed Disk Dynamics

5.1 Aeroelasticity

Aeroelasticity [45] studies the interaction between the fluid and the blade, in the context of this thesis. This interaction consists of a feedback loop in which the pressure field exerted by the fluid on the blade excites a specific modal response, which in turn alters the pressure field, possibly resulting in flutter phenomena that may lead to catastrophic events.
To simplify the complicated three-dimensional flow through the blades in a turbo-engine, it is possible to resort to a two-dimensional strip model, thus neglecting the blade thickness. This model is also commonly used in wing aeroelasticity.
To derive the governing equation, Lagrange's equations [45] are written for a generic variable θ and i-th degree-of-freedom (dof):

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\theta}_i}\right) + \frac{\partial V}{\partial \theta_i} - \frac{\partial T}{\partial \theta_i} + \frac{\partial F}{\partial \dot{\theta}_i} = Q_i\,, \qquad 1 \leq i \leq d \tag{5.1}$$

where T and V represent the kinetic and potential energy, respectively, F is the dissipation function, and Q_i is the generalized force term, i.e. the derivative of the virtual work of the external forces with respect to the virtual displacement. Applying this equation for displacement field, the following expression [45] is obtained, representing the dynamic equation of motion of a d-dof system:

$$[m]\{\ddot{x}\} + [c]\{\dot{x}\} + [k]\{x\} = \{F_A(t)\} + \{F_M(t)\} \tag{5.2}$$

where $\{x\}$ is the vector of the global degrees of freedom of the system. In particular, it collects the structural displacements, generally along the three spatial directions, and possibly also rotational degrees of freedom depending on the adopted model. The structural mass, damping, and stiffness are represented by the matrices $[m]$, $[c]$, and $[k]$, respectively. The forcing terms on the right-hand side represent the external forces due to aerodynamic $\{F_A(t)\}$ and mechanical $\{F_M(t)\}$ sources.
Neglecting forces and damping, the homogeneous form is obtained:

$$[m]\{\ddot{x}\} + [k]\{x\} = \{0\} \tag{5.3}$$

By inserting a trial solution in the form:

$$\{x\} = \{\bar{x}\}e^{i\omega t} \tag{5.4}$$

where ω is the angular frequency, while $\{\bar{x}\}$ is a complex amplitude vector, we obtain:

$$\left([k] - \omega^2[m]\right)\{\bar{x}\} = \{0\} \tag{5.5}$$

This equation is restricted to a single blade and must be solved as an eigenvalue problem.

5.2 Bladed Disk Dynamics

Nevertheless, bladed disks consist of a cylindrical body rotating about a central axis and are composed of N identical sectors that repeat periodically around the circumference. This periodicity allows the dynamic behaviour of the entire assembly to be investigated by analysing a single fundamental sector.
At the disk interfaces, each sector interacts with the two adjacent sectors; these connections are modelled through a coupling matrix $[K_c]$. By considering the adjacent sectors, a full 360° model can be constructed by replicating the single-sector model N times and enforcing the coupling conditions between neighbouring sectors. Therefore, if d denotes the number of dof of one sector, the total number of degrees of freedom of the bladed disk is $N \cdot d$. This yields a system of $N \cdot d$ equations; in implicit form:

$$[M]\{\ddot{X}\} + [K]\{X\} = \{0\} \tag{5.6}$$

Here, $[M]$ is a diagonal matrix whose (n, n)-th block contains the mass matrix of the n-th fundamental sector; conversely, $[K]$ is a block-circulant matrix whose n-th row contains the stiffness matrix of the n-th fundamental sector together with the coupling matrices to the adjacent sectors. Using a solver, the eigenvalue problem can be solved, yielding $N \cdot d$ eigenvalues, representing the natural frequencies, and same number of eigenvectors, which are the mode shapes of the bladed disk.

5.2.1 Cyclic Symmetry

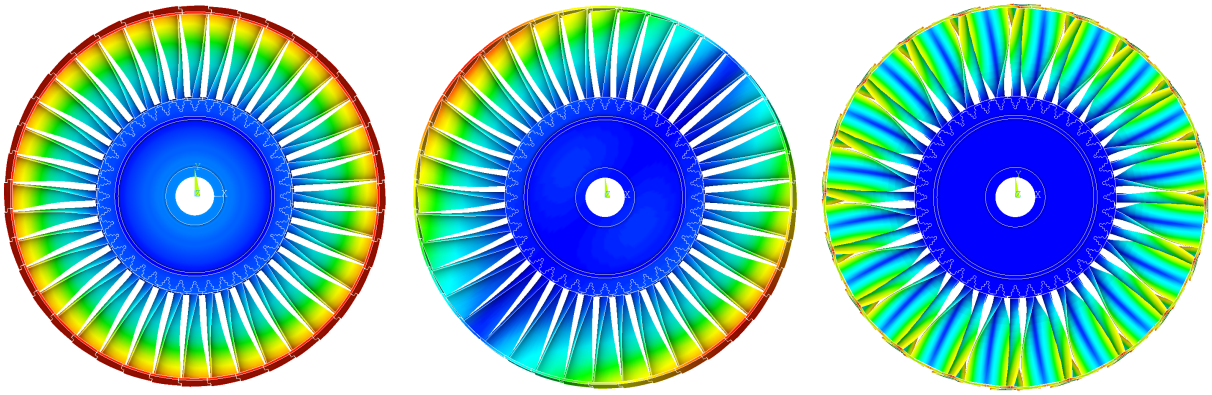
Because computing the eigenvalues of these large matrices is computationally expensive, the cyclic symmetry assumption is employed. This reduces the problem to the analysis of a single sector with d degrees of freedom, from which the eigenvectors of the entire bladed disk, i.e. of all N sectors, can be reconstructed.
At the basis of this approach, certain theorems confirm the existence of two complex mode shapes, $\bar{\Theta}_+$ and $\bar{\Theta}_-$, associated with a natural angular frequency ω_n. These eigenvectors are mutually orthogonal and complex conjugate, and a linear combination of Φ and $\hat{\Phi}$ generates the mode shapes of each sector of the disk. In other words, $\bar{\Theta}_+$ and $\bar{\Theta}_-$ form a basis for the modal shape subspace. It can be shown that the mode shapes of the n-th sector are related to those of the neighbouring sectors through the definition of the Inter-Blade Phase Angle φ:

$$\begin{cases} \bar{\Theta}_+^{(n-1)} = \bar{\Theta}_+^{(n)} e^{-i\varphi} \\ \bar{\Theta}_+^{(n+1)} = \bar{\Theta}_+^{(n)} e^{+i\varphi} \end{cases} \qquad \begin{cases} \bar{\Theta}_-^{(n-1)} = \bar{\Theta}_-^{(n)} e^{+i\varphi} \\ \bar{\Theta}_-^{(n+1)} = \bar{\Theta}_-^{(n)} e^{-i\varphi} \end{cases} \tag{5.7}$$

This means that it is possible to reconstruct the modal shape of the entire disk by solving only a sector with d degrees of freedom.
In other words, φ represents the phase shift of the modal shape between neighbouring blades. After a rotation through N sectors, the eigenvector of sector 1 must be recovered. Therefore, rotating through all N sectors, the angle φ must correspond to an integer number of full revolutions. It follows that:

$$\varphi = ND \cdot \frac{2\pi}{N} \tag{5.8}$$

where ND is the nodal diameter. From a physical point of view, it represents the number of nodal diameters, i.e. the loci of nodes of the disk where the modal displacement is zero.



ND = 0 ND = 1 ND = 20

Figure 5.1: Mode shapes corresponding to different nodal diameters[1].

Finally, it is possible to compute the physical displacement field of a given sector. In fact, assuming a solution in the form introduced in Eq.5.4, one has:

$$\{X\} = Re\left(\{\bar{X}\}e^{i\omega t}\right) = Re\left(\{\bar{\Theta}\}e^{i\omega t}\right) \tag{5.9}$$

Moreover, by substituting Eq.5.7 into the previous expression, it follows that $\bar{\Theta}_+$ and $\bar{\Theta}_-$ are counter-rotating waves.

5.2.2 FreND

Frequency and nodal diameter values can be plotted on the vertical and horizontal axes, respectively, in a diagram known as the FreND diagram. It is possible to visualize modal families, which highlight the evolution of the mode shapes as the nodal diameter varies.

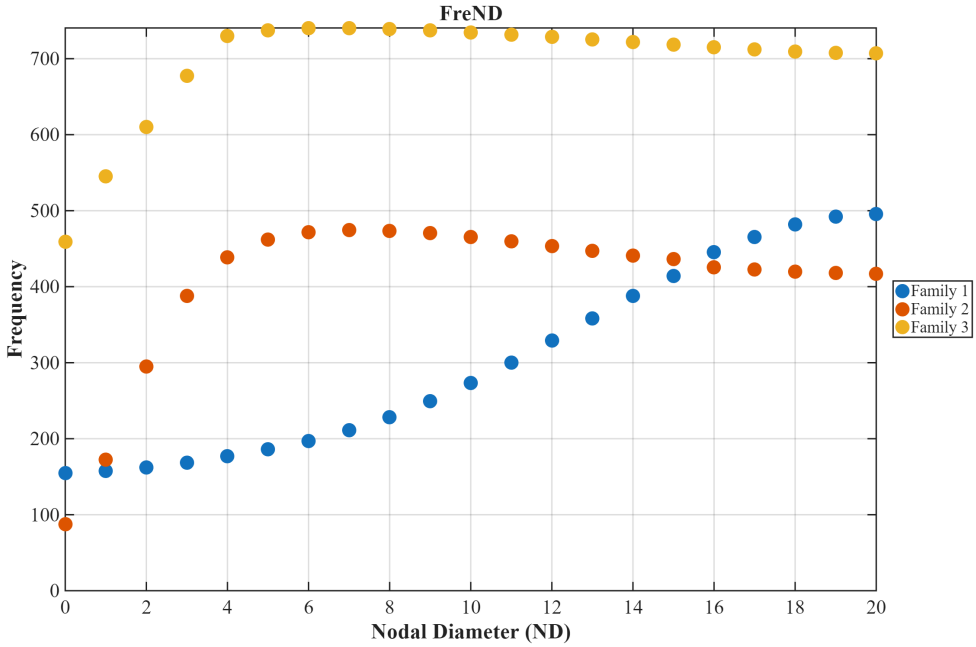
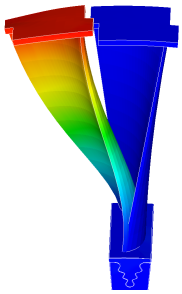
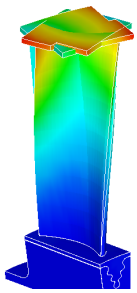
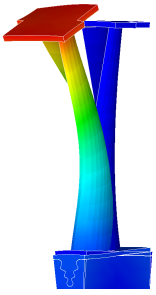
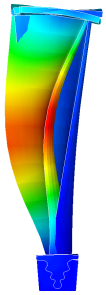


Figure 5.2: FreND.

[1]Source of the model: material provided in the Rotordynamics course by Prof. Stefano Zucca, *Politecnico di Torino*

5.2.3 Modal Shape Classification

$\{X\}$ can assume different forms because the corresponding eigenvector varies with frequency and, in general, with low nodal diameter values. This means that different modal patterns can be identified and classified, which is the main objective of this project.
Today, mode shapes are still classified manually according to the qualitative effect of the displacement field on the blade. Since the threshold values adopted for classification may vary from one expert to another, the process is inherently affected by a certain degree of subjectivity. The classification is generally based on the most common, recurrent, and morphologically pure modal families, as illustrated in Table 5.1. In practice, however, mode identification is often complicated by the presence of mixed modes, which may exhibit combined features of different modal families.

Mode	Shape	Description
Edgewise (EW)		Predominantly tangential displacement with respect to the disk. The blade mainly bends in the circumferential direction, with motion developing edgewise rather than along the disk axis.
Twist (TW)		Mode characterized by torsional deformation concentrated near the blade tip, which rotates about an axis approximately parallel to the blade span. This mode is typically found in the same family as the edgewise mode at high nodal diameters.
Flapwise (FL)		Bending mode mainly directed along the disk axis. The blade motion develops in the flapwise direction, i.e. normal to the plane of rotation.
Flexural (F)		Flexural deformation localized around the midspan region of the blade. The displacement resembles a local bulging or bowing of the blade surface, without a clearly defined global bending direction.

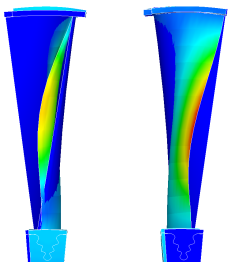
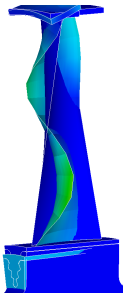
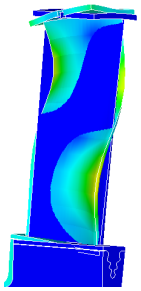
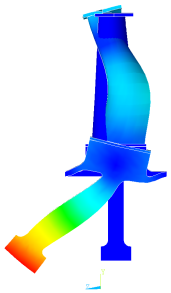
Mode	Shape	Description
Torsional (T)		Torsional deformation involving the leading edge (LE) and trailing edge (TE), mainly localized near the blade mid-span. The LE and TE move out of phase, indicating a rotation of the mid-span section about the blade axis rather than a purely translational motion. Unlike the flexural mode, the section undergoes a local twist.
Second Flexural (2F)		Higher-order flexural mode, typically appearing as a localized planar sinusoidal deformation, often more evident near the trailing edge.
Second Torsional (2T)		Higher-order torsional mode, often recognizable by an "X-shaped" displacement pattern on the wetted blade surface. Each branch of the pattern corresponds to regions of maximum and minimum displacement.
DISK		Modes dominated by disk motion, possibly involving additional components such as the angel wing and seal. These modes typically arise at low nodal diameters and are not treated in the same way as the main blade modal families listed above, since they exhibit less regular and less recurrent characteristics.

Table 5.1: Most recurrent and morphologically pure modal families.

Moreover, for low nodal diameters, the **Rigid** mode may appear. This mode corresponds to a displacement field with no spatial gradients along a given direction, typically the radial direction in the disk reference frame.

6 - BeaML

6.1 Process Overview

`BeaML` is a preliminary project devoted to the classification of beam modal shapes (Beam) through a Machine Learning (ML) code. Its purpose is mainly exploratory, namely to understand whether standard ML techniques could be effectively implemented before addressing the more complex blade case.
From a physical point of view, the beam represents a simplified structural model of the blade. For this reason, it was reasonable to start from such a simplified model.
This experience was developed through a code for database generation, based on the following steps:

- solving the governing beam equations by modeling the structure as a continuous system;
- perturbing the exact solution by altering the final position of the nodes;
- computing the required physical features;
- creating a structured table for ML training.

6.1.1 Euler-Bernoulli Model

The Euler-Bernoulli equation for the continuous modeling of beams is written as follows:

$$V(z) = A\cos(\alpha z) + B\sin(\alpha z) + C\cosh(\alpha z) + D\sinh(\alpha z) \in \mathbb{R}^{n\times 1}\,, \quad \alpha^4 = \frac{\mu\omega^2}{EI_\theta} \tag{6.1}$$

where z is the beam axis, A, B, C, and D are coefficients, μ is the mass per unit length, ω is the angular frequency, E is Young's modulus, and I_θ is the area moment of inertia, with $\theta = x$ if the mode lies in the yz plane and $\theta = y$ if the mode lies in the xz plane. Finally, n denotes the number of nodes long the beam length. The parameter α is the non-dimensionalized frequency parameter, obtained by solving the corresponding nonlinear characteristic equation:

$$cos(\alpha L) = -\frac{1}{cosh(\alpha L)} \tag{6.2}$$

Similar process are valid for both torsional and axial modal shapes.

6.1.2 Data Generation Approach

At this stage, it is necessary to create a cosistent dataset for training and validation. Since no similar dataset are available, it is possible to define a perturbed nodal position vector P using as reference the exact modal shape $\{V\}$ and defining a displacement perturbation vector δ such that:

$$[P] = \{\delta\} \cdot \{V\}^T \in \mathbb{R}^{3\times n} \tag{6.3}$$

Here, $\{\delta\}$ is a three-component perturbation vector with one dominant component. From a physical point of view, the perturbation is dominant in the plane in which the original mode lies, whereas smaller perturbations are introduced along the other directions. By varying the intensity of the components of δ, different configurations can be generated, as shown in Fig. 6.1.

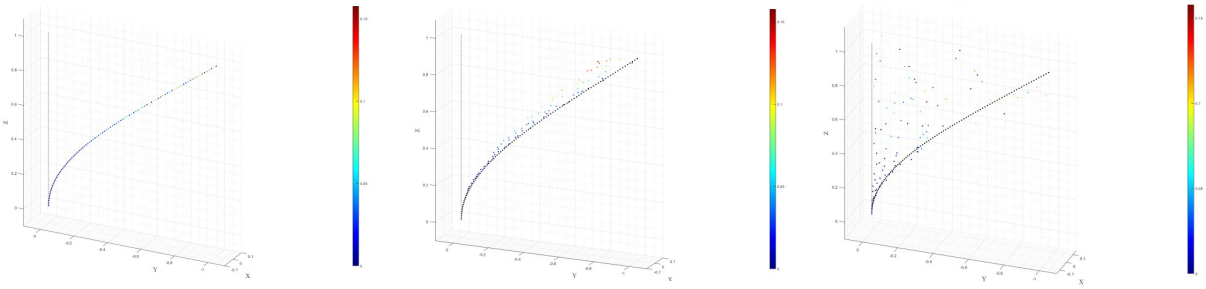


Figure 6.1: Comparison between the exact modal shape and different levels of perturbation.

Since machine learning operates on features, it is necessary to extract meaningful physical parameters. A dedicated code therefore fits each modal shape by least squares using the *Levenberg–Marquardt* algorithm in robust form, taking Eq. (6.1) as the reference model, and extracting the coefficients A, B, C, and D, together with the perturbed frequency parameter $\alpha^{(p)}$. In addition, the modal energy $k_\theta = u_\theta^2$ is computed, where u is the displacement with respect to the unperturbed condition, together with the number of modal nodes, i.e., the nodes where the displacement is approximately zero.
The labeling procedure, namely the output vector, is represented by a single string of the form ABCC, where A identifies the modal type (1 = flexural, 2 = torsional, 3 = axial), B identifies the direction (0 for torsional and axial modes, 1 if the mode lies in the xz plane, 2 if it lies in the yz plane), and CC identifies the modal index, which is physically related to the number of modal nodes.

6.1.3 Training

Training is performed through a specific app, which allows the structured table of physical features to be imported and all candidate models to be trained in parallel. At the end, the best model can be selected either according to the training time or according to the achieved accuracy. Since the training times were generally low, an accuracy-based selection criterion was adopted. In this case, the most accurate model was an Ensemble Bagged Trees classifier.
The strongly diagonal structure of the training confusion matrix could suggest overfitting. However, the accuracy obtained during validation suggests the opposite, indicating instead a very good generalization capability for the considered problem.

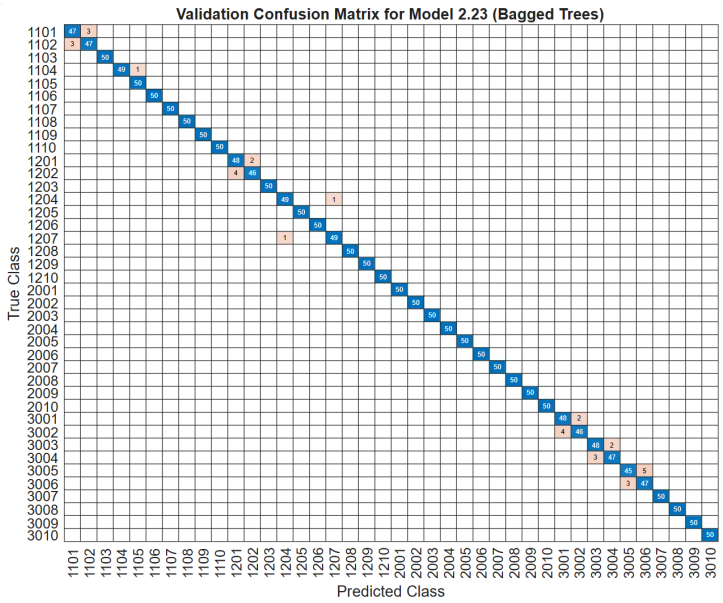


Figure 6.2: Training confusion matrix.

6.1.4 Testing

The most interesting phase is the testing one. A macro in a commercial software was written to generate beam models whose elements are characterized by different geometrical and material properties from one element to another. Although this does not have a direct physical counterpart, it is useful for assessing the potential of ML methods. The variation is essentially governed by a parameter called `randomness`, expressed as a percentage, which measures the degree of variation in geometric dimensions and Young's modulus between adjacent elements. As this parameter increases, spikes may appear in the results and alter the model prediction.

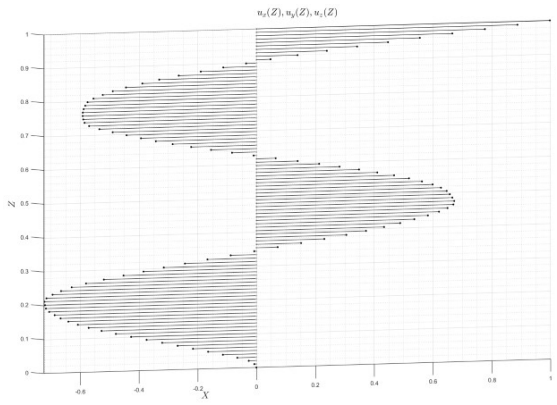


Figure 6.3: `randomness` = 1.5%

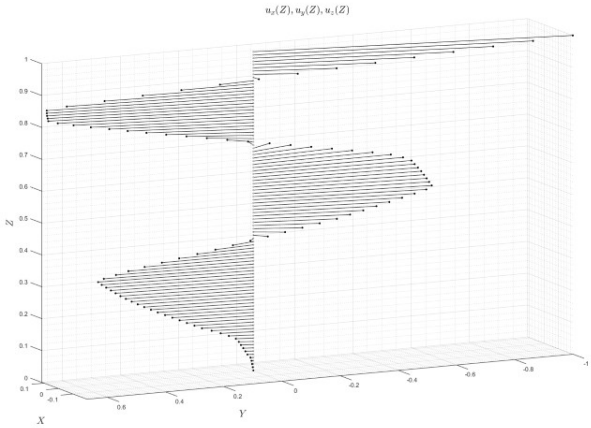


Figure 6.4: `randomness` = 50%

Figure 6.5: Comparison between different levels of `randomness`.

The displacement values are then imported and the physical parameters required by the code are extracted. Finally, a weighted criterion is defined in order to evaluate the predicted classes. In particular, a weighted mark is introduced to distinguish it from the scores directly produced by the classifier. This mark mainly rewards the correct identification of the modal type, assigns a smaller weight to the direction, and only weakly penalizes small differences in the modal index with respect to the true class. In this way, true labels, marks, and scores can be compared graphically.

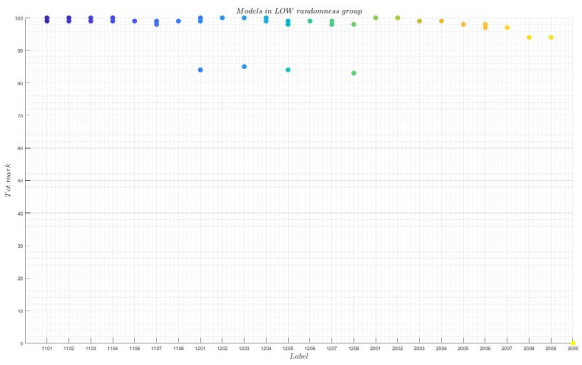


Figure 6.6: Marks vs classes.

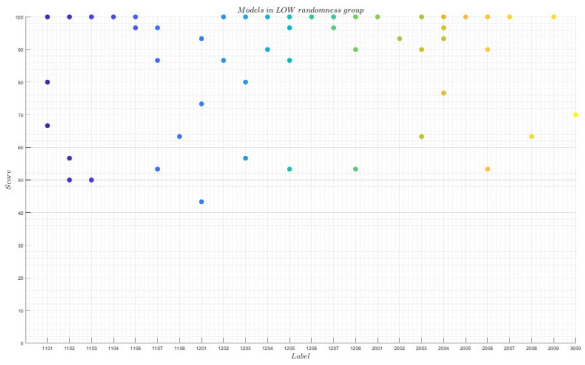


Figure 6.7: Scores vs classes.

From this representation of the testing CM, some considerations can be drawn:

- the change in the nature of the inputs, from mathematical (since the perturbation is random and artificially generated) to physical (since it derives from FEM analyses), introduces a certain degree of uncertainty;
- this uncertainty is mainly concentrated in the modal index, as confirmed by the presence of multiple points above the 100% mark. This may be related to the fitting procedure, which can obscure or duplicate some modal nodes, since the perturbation is no longer purely mathematical but physically induced.

Finally, as expected, increasing the level of randomness leads to a reduction in performance, with a corresponding decrease in marks and scores.

6.1.5 Conclusions

In conclusion, this experience is very useful for testing some of the methodologies that are later implemented in the main algorithm, from perturbation and testing to training strategies. Moreover, it can also be used for educational purposes as an interface for the automatic classification of beam modes. Finally, some examples of the outputs produced by the final tool based on Ensemble Bagged Trees are reported below.

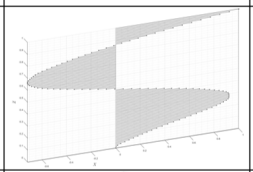
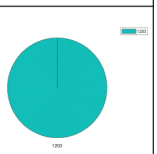
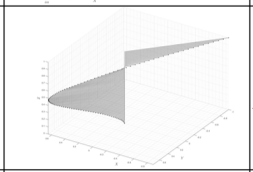
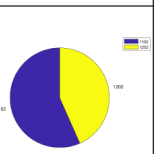
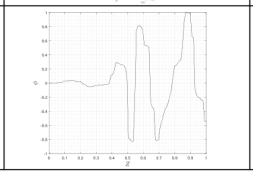
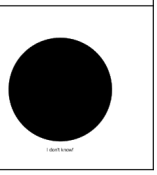
Ease of classification	Example	AI score
Simple		
Difficult		
Impossible		

Figure 6.8: `BeaML` possible outputs.

7 - AIseeU

7.1 Problem Description

As anticipated in Sec. 2.1, the true objective of this thesis is to develop an AI framework capable of literally "seeing" U, where U denotes the modal displacement field, and predicting the correct labels directly from actual blade FEM input data.
The input matrices are contained in the so-called RPT files and exhibit the following characteristics:

- complicated **three-dimensional** displacement fields, since blades may exhibit intrinsically three-dimensional geometries and stagger angles that generate more complex vibration modes;
- **complex-valued** matrices;
- **non-unique** mathematical representations, in the sense that the eigenvectors may vary in their numerical entries while preserving the same overall modal shape; as a consequence, different numerical values may still correspond to the same label;
- **limited** amount of data available;
- physical and geometrical **features** that are too difficult to compute or select by means of simple handcrafted criteria;
- frequency and ND do not provide sufficiently informative features to be included directly in the dataset, since different engines, or even different stages of the same engine, may lead to significantly different FreND representations for analogous modal behaviors.

7.1.1 FreND Analysis

An important preliminary step before starting the training process is to assess whether useful information can be extracted from the available data. In the present case, an attempt was made to identify recurring patterns in the mode positions within the FreND diagram, with the aim of improving mode prediction. However, the outcome was negative. Despite applying several normalization strategies with respect to different reference entities, and with the objective of making the FreND representation as comparable as possible across the available cases, no clear evidence of exploitable information was found for training purposes. The normalization approaches adopted are summarized in Table 7.1.
Here, $F = \{1, 2, 3, 4, 5\}$ denotes the modal family index, whereas $S = \{1, 2, 3\}$ denotes the stage index.

Formula	Meaning
$(\omega^\star, ND^\star)\|_S = \dfrac{(\omega, ND)\|_S}{\max\left[(\omega, ND)\|_S\right]}$	Used to evaluate the entire FreND diagram of a single stage, including all modal families, normalized with respect to the maximum frequency and ND values of that same stage.
$(\omega^\star, ND^\star)^{local}\Big\|_{F,S} = \dfrac{(\omega, ND)\|_{F,S}}{\max\left[(\omega, ND)\|_{F,S}\right]}$	Used to evaluate the trend of a single modal family, normalized with respect to the maximum values of that same modal family within the analyzed stage.
$(\omega^\star, ND^\star)^{global}\Big\|_{F,S} = \dfrac{(\omega, ND)\|_{F,S}}{\max\left[(\omega, ND)\|_{\forall S}\right]_F}$	Used to evaluate the trend of a single modal family, normalized with respect to the global maximum of that modal family across all analyzed stages.

Table 7.1: Normalization strategies adopted for FreND analysis.

7.1.2 Data Inspection

A further inspection step was introduced in order to verify the consistency of the processed modal data and assess whether the extracted representations were sufficiently informative for the subsequent learning phase.

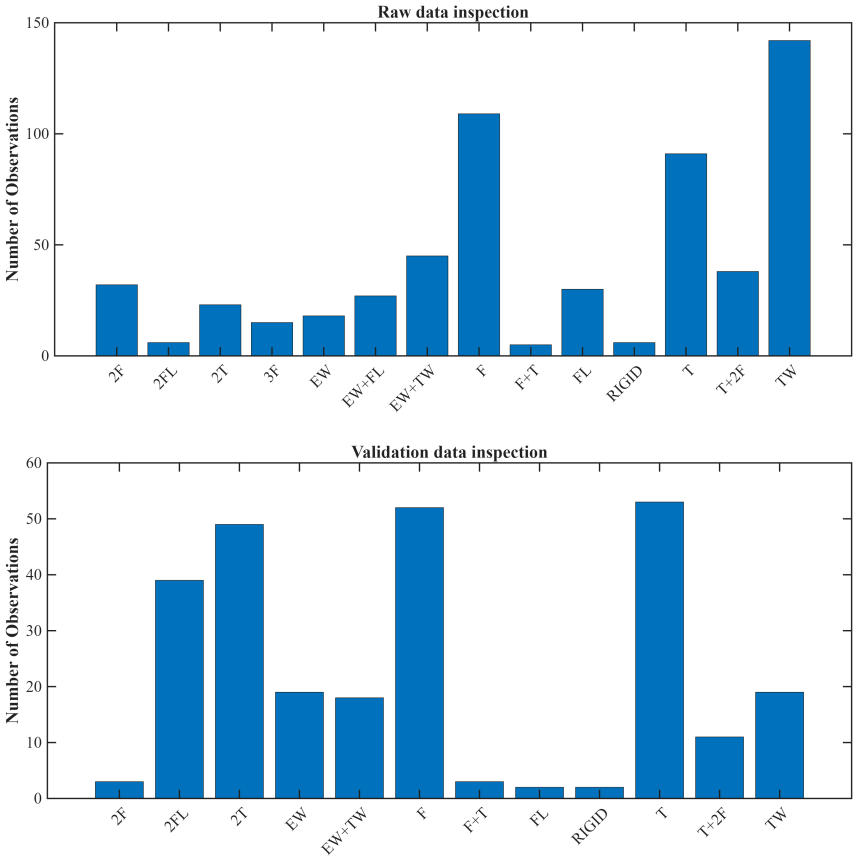


Figure 7.1: Datasets inspection.

7.2 Process Description

To address the issues illustrated in Sec. 7.1, two dedicated functions were developed to solve most of the difficulties related to geometry, meshing, and complex-valued inputs. In addition to the standard phases of training, validation, and testing, the limited amount of data available required the application of an oversampling strategy in order to enlarge the input dataset and improve the robustness of the model.
During training, the weights of all convolutional layers were computed and stored in a dedicated structure, together with the other relevant model parameters and the CNN architecture, so that the trained model could be reused for subsequent computations.
Furthermore, during the validation phase, the decision threshold maximizing the F1-score was determined.
To provide a meaningful contribution to the aerospace field, data from in-service engines are selected for the training and validation phases, whereas testing and the first practical application of the model are carried out on an engine concept for which no FreND diagram is available. In particular, the datasets corresponding to the most representative nodal diameter values are used to construct the training set, denoted as `X_train`. By contrast, the validation phase is performed using fitted-ND data, collected in `X_validation`, in order to assess the behavior of the model in transition regions between different modal families, even though such transitions are not explicitly included in the training set.
This entire procedure was repeated for different combinations of learning rate, number of convolutional layers, latent channels, activation function, and convergence boosters, in order to identify the configuration yielding the best overall performance.
The difficulties motivating the adoption of these specific processing and modeling solutions are summarized in Table 7.2.

Problem	**Adopted solution**
Complicated three-dimensional displacement fields.	Implementation of a dedicated pre-processing routine, namely `mode_reader.m`, to map the blade onto a simplified geometry and obtain a standard representation of the modal shape displacement.
Non-unique mathematical representations of the same modal shape and complex-valued inputs.	Creation of a processing function, namely `freeze_mode.m`, to extract most representative features map and make modal shapes more comparable.
Limited amount of data available.	Use of an oversampling strategy together with the construction of specific training and validation datasets in order to improve robustness and generalization capability.
Physical and geometrical features too difficult to extract manually.	Adoption of a 2D CNN, stored in `cnn.m` architecture, allowing the model to learn relevant patterns automatically from the processed modal representation.
Frequency and ND not sufficiently informative as direct features.	Removal of FreND quantities from the training and validation datasets, restricting their role to visualization and label-assignment support.

Table 7.2: Main problems addressed in this work and corresponding adopted solutions.

7.2.1 Tool Overview

The tool takes the RPT files as input and produces a labeled FreND diagram as output. To address the geometrical and representational difficulties of the raw FEM input, the functions `mode_reader.m` and `freeze_mode.m` are applied upstream of the 2D CNN evaluation.
The following quantities are defined and used consistently throughout the training, validation, and testing phases:

- $X_{\text{raw}} \equiv \bar{\Theta}$: the complex eigenvector matrix extracted from the RPT files;
- $X_{\text{in}} \in \mathbb{C}$: the dataset preprocessed by `mode_reader.m`;
- $X \in \mathbb{R}$: the dataset further processed by `freeze_mode.m`, corresponding to the inputs used in `X_train` and `X_validation` variabiles.

To address the difficulty of manually defining robust physical features, a 2D CNN was developed so that the model could automatically identify the relevant patterns from the processed inputs.
In addition, the function `frend.m` was implemented to generate the FreND diagram and assign the corresponding labels correctly. The overall structure of the tool is therefore outlined as follows.

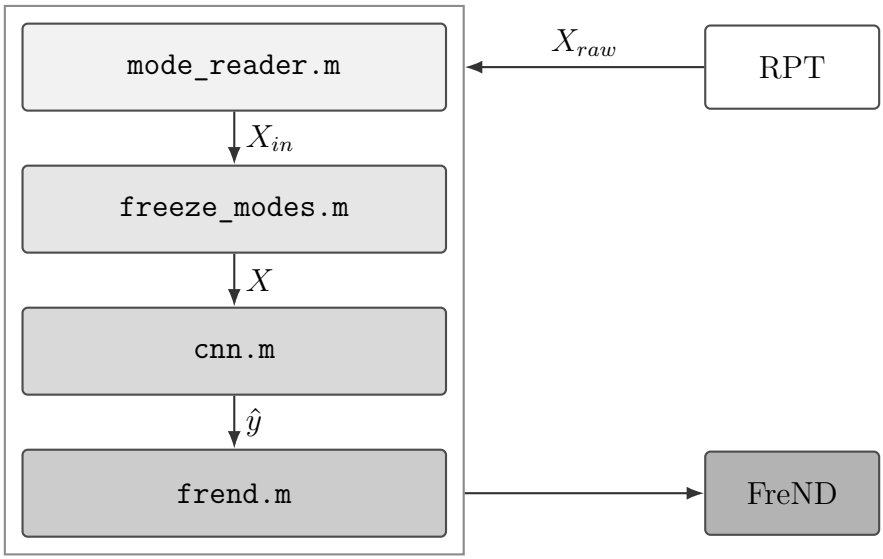


Figure 7.2: Tool structure

The main functions composing the tool are described in the following sections.

7.3 Data Generation

Data generation is a necessary step to build a consistent dataset containing only meaningful information, thereby reducing the risk of overfitting.
First, the functions `mode_reader.m` and `freeze_mode.m` are applied to construct the maps associated with the original modal shapes. Subsequently, a detailed labeling procedure is carried out to generate the target sets `Y_train` and `Y_validation`.
Then, in order to preserve the characteristics of the original modal displacements, synthetic inputs are generated by means of a custom SMOTE-based interpolation approach within a hybrid oversampling and undersampling strategy. This method also includes physically consistent perturbations of the synthetic mode shapes.

7.3.1 `mode_reader.m`

The `mode_reader` function performs several tasks at different stages of the pre-processing pipeline:

- **reading** the RPT files and extracting the complex displacement fields X_{raw};
- reading and post-processing the `t_mesh` file, which contains the connectivity matrix and the spatial coordinates of the blade nodes;
- removing the **shroud** and the **blade root** (as in Fig.7.3) by applying a threshold based on the derivative of the cross-sectional area along the blade axis, thus generating the matrices X_{in};
- plotting the real mode shape on the actual blade geometry over the domain $[0, 2\pi]$;
- performing the **plate mapping** procedure, which consists in transforming the real displacement field into 1×1 plates with a regular, homogeneous, and structured square grid, thus mapping the data from the physical domain to a mathematical one;
- plotting the real mode shape on the plate over the domain $[0, 2\pi]$.

It should be noted that `mode_reader` does not modify the eigenvectors themselves, but only preprocesses the data in order to retain the wetted part of the blade.

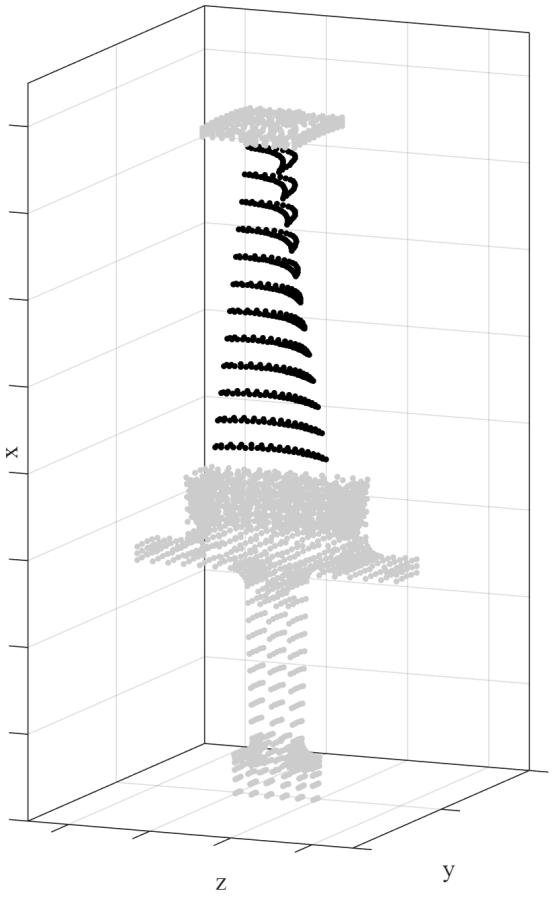


Figure 7.3: Wetted blade isolation[1].

The mapping procedure is carried out by replacing the original three-dimensional geometry with two **normalized surface coordinates**, and by repeating the same construction for the pressure and suction sides.
The first coordinate, denoted by ξ, represents the chordwise position and is directly obtained from the local chord fraction. The second coordinate, denoted by η, represents the spanwise position and is computed by normalizing the span coordinate with respect to the minimum and maximum accepted span values. Both coordinate variables are discretized using $n = 100$ nodes.
As a consequence, each valid node of the blade is projected from its original curved three-dimensional position onto a regular domain of size 1×1, where $\xi = 0$ and $\xi = 1$ identify the chordwise boundaries associated with the leading- and trailing-edge envelope, while $\eta = 0$ and $\eta = 1$ correspond to the root and tip, respectively.
The tensor extracted from the RPT files is denoted by X_{raw}. Then, `mode_reader` performs a cyclic transformation, obtaining a variation over a non-temporal phase domain, as shown below:

$$X_\theta = X_{in} \cdot e^{i\theta}, \qquad \theta \in [0, 2\pi] \tag{7.1}$$

[1]Source of the model: material provided in the Rotordynamics course by Prof. Stefano Zucca, *Politecnico di Torino*.

7.3.2 `freeze_mode.m`

The function `freeze_mode.m` is introduced to address two main issues: the lack of a unique numerical representation for a given modal shape, and the need to handle complex-valued displacement fields in a form suitable for training.
With regard to the first aspect, the same modal shape may correspond to different mathematical representations, since the complex eigenvector can be observed at different phase positions over the cycle domain. For this reason, `freeze_mode.m` analyzes the interval $[0, 2\pi]$ in a discretized manner and extracts the frame corresponding to the **maximum real displacement**, thus enforcing a more consistent and representative description of the considered mode shape. In formulae, the following quantity is considered:

$$X = Re(X_{\theta_{max}}) = Re(X_{in} \cdot e^{i\theta_{max}}) = Re(X_{in}) \cdot \cos\theta_{max} - Im(X_{in}) \cdot \sin\theta_{max} \tag{7.2}$$

where $\theta_{max} = \theta\Big|_{\max(X_\theta)}$. With respect to this value, a sample-wise normalization is then applied. This operation generally improves the convergence of neural networks, since it prevents excessively large value excursions that could make the training process less stable.
A second relevant issue concerns the treatment of the complex-valued nature of the modal displacement field. The obtained results show that the imaginary part of the complex displacement, $Im(X_{\theta_{max}})$, does not provide additional information with respect to the real part for the purpose of modal shape classification, as shown in Fi.7.5. In other words, the imaginary contribution appears to preserve essentially the same spatial pattern as the real one, so that including it explicitly in the training process does not improve the classification results. This is also supported mathematically, since the real and imaginary parts are simply shifted by $\pi/2$:

$$Im(X_{in}e^{i\theta}) = Re(X_{in}e^{i(\theta+\frac{\pi}{2})}) \tag{7.3}$$

This may be justified by the fact that mode recognition is effectively performed on the real modal shape rather than on its imaginary part.
A further attempt was made by considering amplitude and phase maps, which remain constant over the cycle domain. However, after several tests based on different functions, that return phase angle and the arctan, used either with or without the common jump in phase value, it emerged that the phase does not have a clear physical meaning and that its patterns are not repetitive. This suggests that it cannot be reliably used for recognition purposes. Therefore, in the present work, the imaginary part is not included in the final training representation.
Ultimately, the map characterized by the maximum real displacement proved to be the most effective representation for training the model, since the imaginary part does not contribute to pattern recognition, the phase is not unique across samples sharing the same label, and the amplitude does not provide information about the sign of the displacement map, resulting in an interpretation issue for opposite displacement peaks, as in **T** modal shapes.
It should nevertheless be emphasized that this is an empirical result obtained within the present framework. In principle, there may exist a more suitable transformation or a different physical interpretation capable of exploiting the imaginary component in the training process, although such a formulation was not identified in this work.
The second aspect concerns the fact that the pressure side and suction side do not exhibit major differences, which suggests the absence of significant deformation in the direction normal to the blade surface.

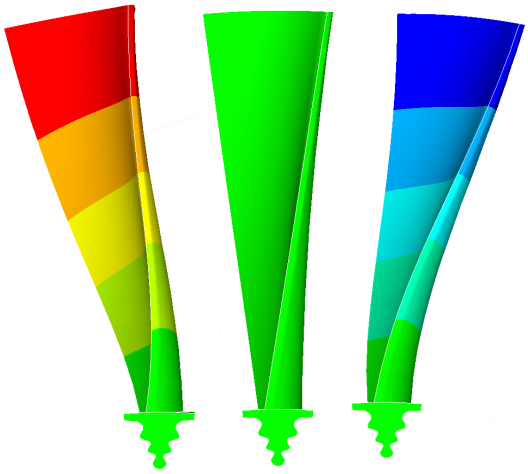


Figure 7.4: Representation of EW modal shape evolution.

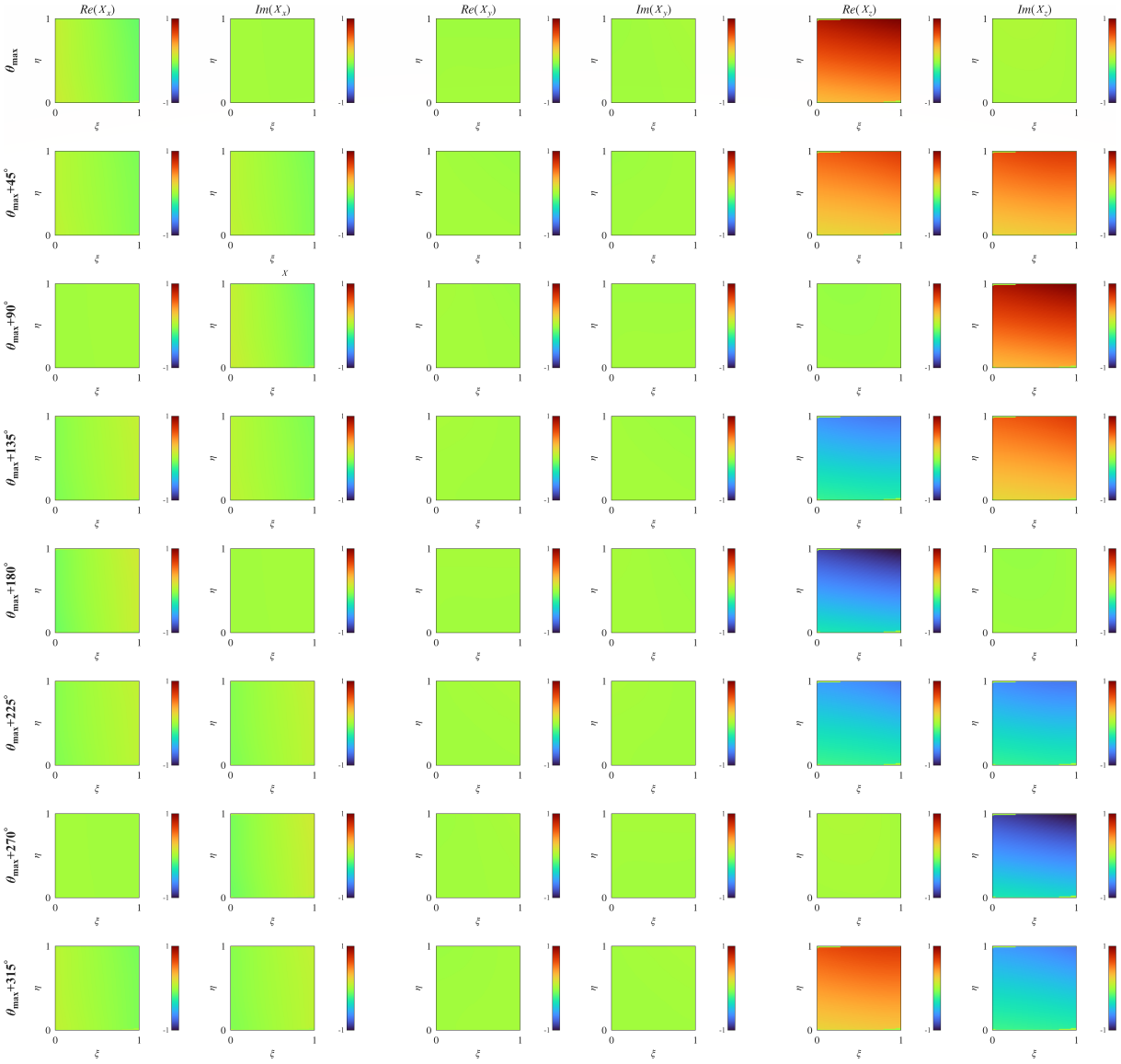


Figure 7.5: EW complex displacement parts maps varying θ starting from θ_{max}, normalized with maximum of each part.

7.3.3 Mode-shape Perturbation Strategy

To enrich the training dataset, each complex modal shape is copied and transformed into a perturbed sample through a multi-stage augmentation procedure. Accordingly, oversampling is applied to rare modes, whereas undersampling is applied to common modes.
Starting from an original sample, the method combines SMOTE-based interpolation among samples belonging to the same class, contamination from different classes opportunely scaled so as not to mislead the training process, a pixelation effect, and a smooth spatial Gaussian field.
Let the original input dataset be denoted as:

$$\mathbf{X}^{(0)} \in \mathbb{R}^{n\times n\times 3\times n_s} \tag{7.4}$$

where n denotes the number of plate nodes, and n_s is the total number of samples. Let y_i denote the class label associated with the i-th reference sample, as usual.

Same-label Interpolation The first augmentation step generates an initial synthetic sample by combining mode shapes belonging to the same class as the reference sample. Consider the generic i-th mode to be replicated, and let $\mathcal{S}$ denote the subset of the training set $\mathcal{T}$ containing only modal shapes with the same label. Among these, at most K_s mode shapes with the same label are included in the interpolation process:

$$\mathcal{S}(y_i) = \{k \in \mathcal{T} \mid y_k = y_i\} \ , \quad |\mathcal{S}(y_i)| \leq K_s \tag{7.5}$$

Then, one sample is randomly selected as the dominant contribution and assigned a weight $\alpha_d \in [0.6, 0.8]$. The remaining samples receive random normalized weights such that the total sum remains equal to one. The resulting mixture is written as

$$X^{(1)} = \sum_{k\in\mathcal{S}(y_i)} \sigma_k \, w_k \, X_k^{(0)} \tag{7.6}$$

where w_k are the normalized weights and σ_k is a sign function:

$$\sigma_k = \begin{cases} 1, & \text{for the dominant sample} \\ \pm 1, & \text{for the remaining samples} \end{cases} \tag{7.7}$$

In this way, the generated sample remains predominantly representative of the original class while still introducing controlled variability.
If only one sample is available in the class, the processed sample is simply taken equal to the reference one:

$$X^{(1)} \equiv X^{(0)} \tag{7.8}$$

Cross-label Mixing To further enrich the perturbation process with a physically based pattern, an additional contribution from samples with different labels can also be included. Let $\mathcal{O}$ denote the subset of the training set $\mathcal{T}$ containing only modal shapes with labels different from that of the i-th mode. Among these, at most K_o samples are considered for cross-label mixing:

$$\mathcal{O}(y_i) = \{k \in \mathcal{T} \mid y_k \neq y_i\} \ , \qquad |\mathcal{O}(y_i)| \leq K_o \tag{7.9}$$

Analogously, a subset of these samples is randomly selected and combined into an auxiliary perturbation term:

$$X^{(o)} = \sum_{k \in \mathcal{O}(y_i)} \sigma_k \, w_k \, X_k^{(0)} \tag{7.10}$$

where, again, w_k are normalized weights and σ_k is the sign function.
This term is then scaled through a positive parameter $A_o > 0$ and modulated by the mean value of $X^{(1)}$ along each dimension, so that the perturbation remains consistent with the amplitude distribution of the processed sample. Therefore, the updated variable is written as:

$$X^{(2)} = X^{(1)} + \frac{1}{A_o} X^{(o)} \cdot \mu\Big(X^{(1)}\Big) \tag{7.11}$$

where $\mu\Big(X^{(1)}\Big)$ denotes the mean value of $X^{(1)}$ computed along each dimension. In other words, information from other labels is added to the amplitude maps in a controlled manner.
The parameter $A_o > 0$ introduces additional variability while keeping its effect under control through a scaling factor, thus avoiding excessive confusion for the model. The additional modulation by $\mu(X^{(1)})$ makes the perturbation amplitude depend on the characteristic scale of the processed sample itself, thereby preserving physical consistency in the generated mode shape. Its optimal value was determined through several iterative tuning loops. From a physical point of view, this phase is consistent because it implicitly imposes a threshold on the extent to which different modal patterns may be mixed before the resulting shape is no longer recognized as belonging to the original class.

Truncated Random Perturbation A stochastic multiplicative perturbation is then applied channel-wise in order to globally perturb the modal map and, at the same time, introduce localized spikes in the values. Let $T \in \mathbb{Z}$ denote the truncation factor. For each iteration $t = 1, \ldots, T$, a scale parameter is defined as

$$\gamma^{(t)} = 10^t \tag{7.12}$$

Two random perturbation terms are introduced. The first one is global and is denoted by R_1, whereas the second introduces localized spikes and is denoted by R_2:

$$R_1^{(t)} = \frac{1}{\gamma^{(t)}} \left(s_1^{(t)} \, X^{(2)} \, \mu\Big(X^{(0)}\Big) \right) \tag{7.13}$$

$$R_2^{(t)} = \frac{1}{\gamma^{(t)}} \left(s_2^{(t)} \, X^{(2)} \, \mu\Big(X^{(0)}\Big) \right) \tag{7.14}$$

Here, $s_1^{(t)} \in [-1, +1]$ is a random scalar, representing a global superposition of the processed variable with a scaled modification of the same map; by contrast, $s_2^{(t)} \in [-1, +1]^{n \times n \times 3}$ is a random sign matrix, representing the insertion of localized spikes. In both cases, the perturbation terms are further scaled by the mean value of $X^{(0)}$ along each dimension, so that the resulting fluctuations remain consistent with the characteristic amplitude distribution of the original sample. This step therefore introduces both global amplitude fluctuations and local pixel-like perturbations, while progressively reducing their magnitude as t increases.

The final perturbation terms are obtained by summing the truncated contributions over all T iterations:

$$R_j = \sum_{t=1}^{T} R_j^{(t)}, \quad j \in \{1, 2\} \tag{7.15}$$

Finally, the perturbed sample is written as

$$X^{(3)} = X^{(2)} + R_1 + R_2 \tag{7.16}$$

Gaussian Spatial Field Lastly, a Gaussian perturbation is applied. This choice is consistent with the need to introduce a probability-based spatial variability that is not explicitly considered during training. In other words, the CNN may tend to become excessively sensitive to the exact location at which patterns appear. The introduction of Gaussian fields is therefore intended to reduce this sensitivity and improve generalization. For each channel d, the Gaussian perturbation field is expressed as the sum of T_g Gaussian functions foe each direction:

$$G_d(\xi, \eta) = \sum_{q=1}^{T_g} s_{q,d}\, A_{q,d} \exp\left(-\frac{(\xi - \xi 0_{q,d})^2}{2\sigma\xi_{q,d}^2} - \frac{(\eta - \eta 0_{q,d})^2}{2\sigma\eta_{q,d}^2} \right) \tag{7.17}$$

Here, $s_{q,d} \in \{-1, +1\}$ is a random sign, $A_{q,d}$ is the amplitude, $(\xi 0_{q,d}, \eta 0_{q,d})$ is the Gaussian center, and $\sigma\xi_{q,d}$ and $\sigma\eta_{q,d}$ are the standard deviations and measure the spatial spread of the Gaussian field along the two directions ξ and η.

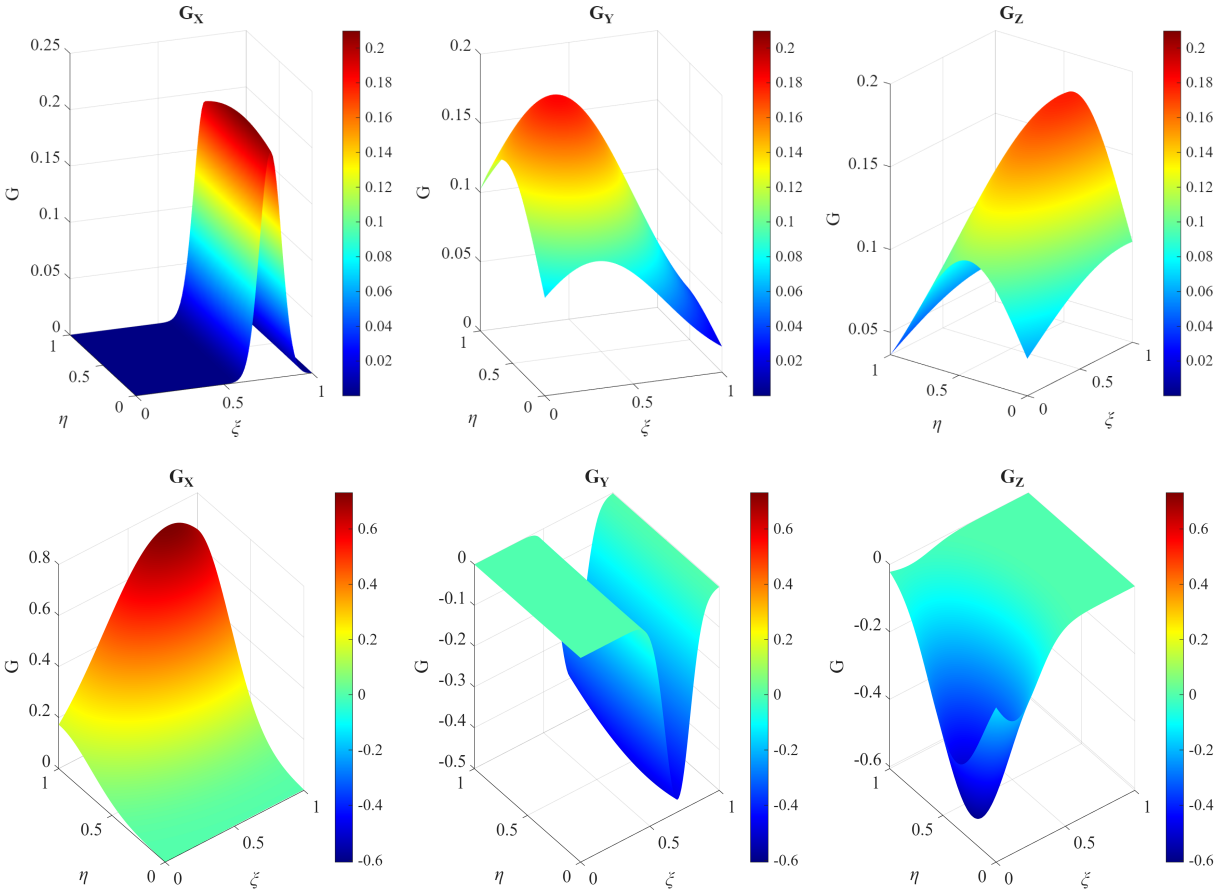


Figure 7.6: Examples of Gaussians applied.

The number of Gaussian terms is chosen so as to generate either a few concentrated perturbations or several lower-amplitude distributed perturbations. In fact, once T_g is fixed, all Gaussian parameters are scaled consistently with this value.

$$T_g = \begin{cases} 1 & if\ T < 1 \\ \text{random integer in } [1, T] & if\ T \geq 1 \end{cases} \tag{7.18}$$

More precisely, for a fixed value of T_g, which represents the number of Gaussian functions being summed, the parameters of each Gaussian term are scaled with respect to it. As a result, when several Gaussian functions are superimposed, each contribution is adjusted so that the overall perturbation remains physically meaningful and does not become excessively strong.
The Gaussian field is then added to the processed sample through:

$$X_d^{(4)} = X_d^{(3)} + \frac{\mu_d}{G_f} G_d \tag{7.19}$$

where $G_f > 0$ controls the intensity of the spatial perturbation, while the mean magnitude of each direction is $\mu(X^{(0)})$.

Final Output Overall, this augmentation strategy preserves the global structure of the original mode shape while increasing intra-class variability through stochastic interpolation and spatial perturbations, as presented in Tab.7.3 with extreme visual exaggeration of the amplitude values. The perturbation chain is applied only to underrepresented labels, whereas the most populated ones are reduced through undersampling. Class balancing is governed by a scaling factor which, multiplied by the maximum class cardinality, defines the target number of samples for each label and thus produces an approximately flat label distribution, as shown in Fig.7.7.
All the parameters introduced in this section were iteratively tuned in order to avoid both overfitting and underfitting. In particular, the best performance was obtained by setting the replication factor with respect to the maximum number of modes equal to 0.8. Overall, the generated samples improve the robustness and generalization capability of the learning model.

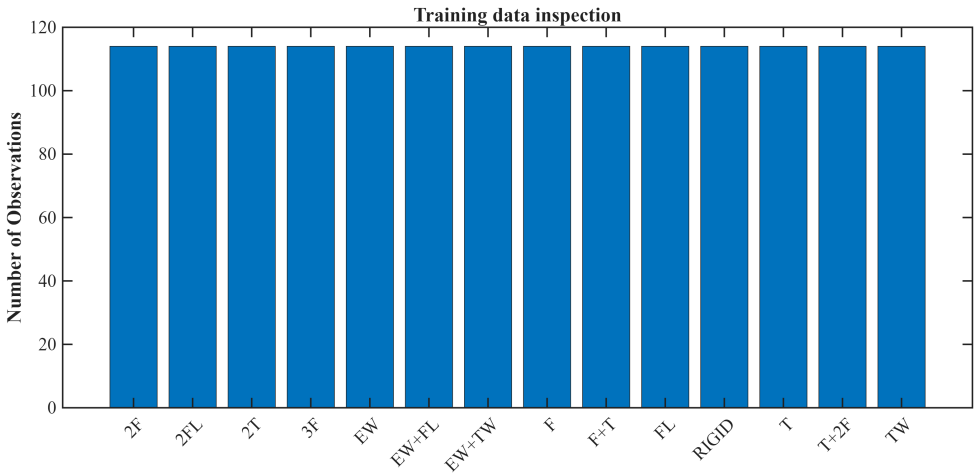


Figure 7.7: Training dataset inspection.

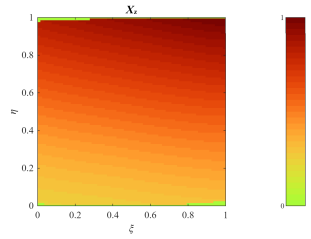
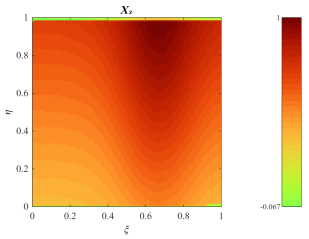
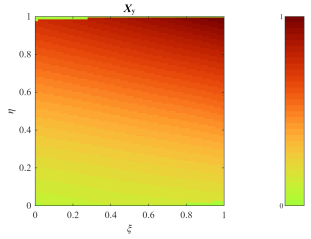
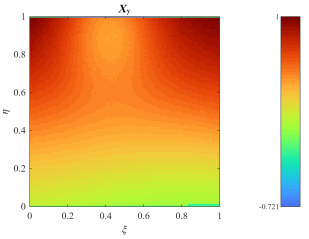
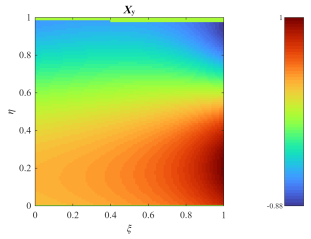
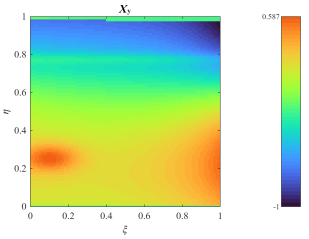
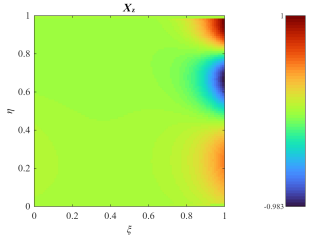
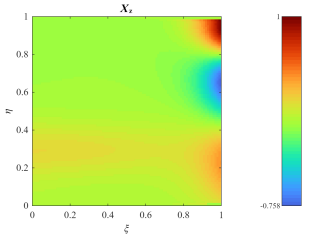
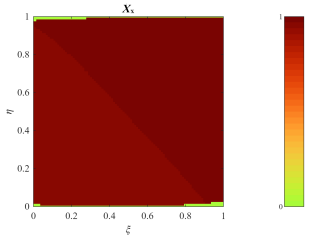
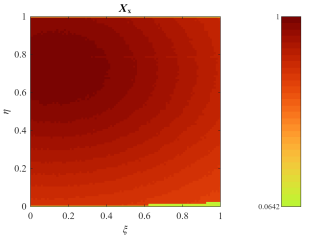
Mode	Original	Perturbed
EW		
FL		
2FL		
3F		
RIGID		

Table 7.3: Comparison between original and perturbed modal-shape representations with exaggerated amplitude values.

7.3.4 Labelling

Labelling is undoubtedly the most delicate part of this thesis. To date, this procedure is typically carried out directly on blade displacement plots by highly experienced users, who are able to identify even very subtle features. For this reason, the assignment of a modal label is inherently subjective.
The first step of this work was therefore to understand, together with domain experts, how this assignment is performed in practice. Once this expert-driven procedure had been clarified, the next objective was to identify geometrical features in the real displacement field capable of justifying and supporting such a choice.

In this respect, this thesis introduces an **important novelty**: the analyses were not performed directly on the blade geometry, but on the corresponding mapped representations. In a sense, the most effective way to predict possible issues from the model is to analyze the same type of input that the model itself receives.
Finally, the conversion from string labels to numerical form was performed using a **true multi-hot encoding** approach, in which each label is represented as a vector composed of 1s and 0s. In this framework, labels behave as boolean indicators that can be either activated or deactivated. Consequently, after the CNN computations, the network outputs must be converted into logits and then interpreted through a thresholding operation.
However, the use of true multi-hot labelling also presents a limitation, namely the lack of physical consistency. In fact, the modes are treated as independent classes that may appear in mixed combinations. As a result, modes with index greater than one are interpreted as entirely distinct from the fundamental ones, e.g. F and 2F, rather than as manifestations of the same spatial pattern repeated over the domain. Therefore, the intrinsic physical relationship among harmonically related modes is not explicitly exploited.
A particularly challenging choice concerned the inclusion in the training database of the so-called **transition** modes, namely cases in which one mode evolves into a different one at low nodal diameters. In the presence of a training dataset that is not sufficiently dense in terms of nodal diameter, it is not possible to clearly distinguish whether both modes are simultaneously active or whether one of them is dominant. In such cases, only the recurrent mixed configurations showing sufficiently distinctive features in the analyzed data were explicitly introduced as mixed labels, namely: EW+TW, T+F, and T+2F, reported in Tab.7.4.
Finally, DISK modes or in general system modes and unclear cases are classified as NC (Not Classified).

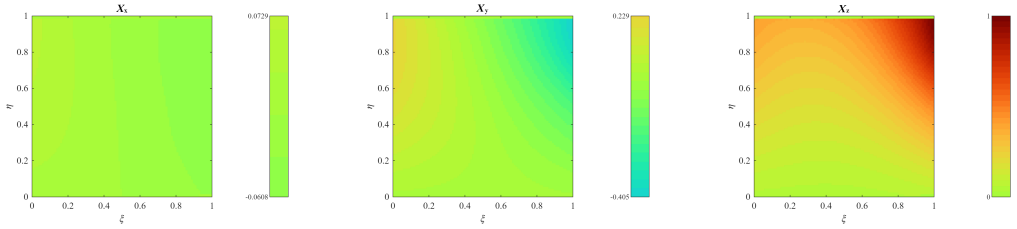
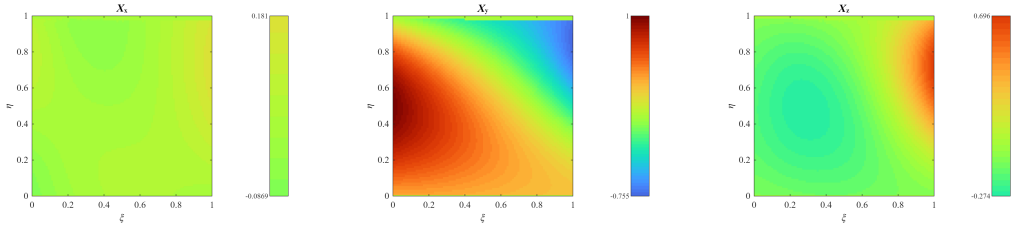
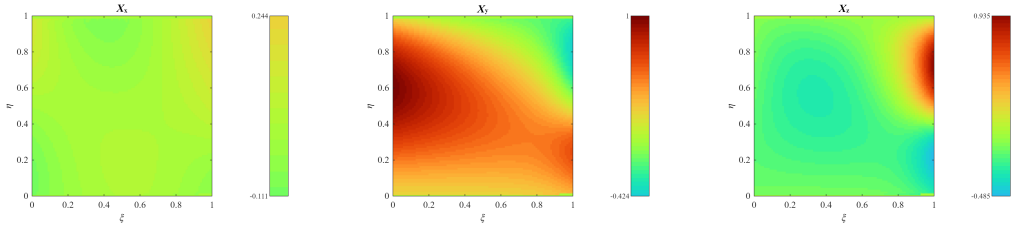
Label	Maps
EW+TW	
T+F	
T+2F	

Table 7.4: Examples of transition-modes.

7.3.5 GNN vs Plates: Computational Cost

Before developing the final CNN-based framework, a preliminary sequence of alternative architectures was explored. The first attempt consisted in implementing a **Graph Convolutional Network** (GCN), so as to process the blade directly on its original discretized domain. Subsequently, the adjacency information was removed, leading to a simplified MLP-like formulation, and finally a convolution-based approach on plate representations was introduced. This last step significantly improved the prediction accuracy, since the convolution operation captures local displacement gradients and spatially correlated patterns rather than relying on isolated nodal values only.
The choice of using maps as input is also well supported in the literature, including applications related to pattern regression and structured-field learning[47]. For this reason, within the present project, it is important to assess whether the construction of the plates is actually advantageous, or whether a GCN applied directly to the blade geometry would have been sufficient.
GCNs are characterized by the following layer-wise propagation rule:

$$H^{(i)} = \alpha^{(i)}\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(i-1)}W^{(i-1)}\right) \tag{7.20}$$

where, according to the adopted notation, $\tilde{A} = A + I_N$ denotes the adjacency matrix including self-connections, $W^{(i-1)}$ is the trainable weight matrix of the $(i-1)$-th layer, and $\alpha^{(i)}$ is the activation function.
According to the formulation proposed in [48], the graph convolution operation can be implemented with a computational cost of order $\mathcal{O}(|E|)$ when sparse matrix operations are adopted, where E denotes the set of graph edges and $|E|$ is the corresponding number of connections between neighboring nodes.
As a consequence, the **computational convenience** of a GCN with respect to a CNN strongly depends on the sparsity of the adjacency matrix. From a purely theoretical point of view, it is therefore not possible to claim a priori that a CNN operating on plate representations is always faster or more effective than a GCN applied directly to the blade. However, in the present case, the adjacency matrix is far from being highly sparse (as described qualitatively in Fig.4.11) because of the large number of nodal connections. Under these conditions, the plate-based approach becomes qualitatively **more advantageous**.
Moreover, what cannot be established purely on theoretical grounds can be assessed empirically. In the present framework, the construction of the plates has a very limited computational cost, since it only requires geometric normalization and domain subdivision, and it is performed once during pre-processing. By contrast, the GCN must repeatedly process the graph structure during training and becomes significantly more demanding when multiple combinations of training parameters have to be tested.
Overall, the plate-based strategy represents an effective compromise to overcome the geometrical complexity of the original blade domain. A GNN-based approach would allow the direct use of blade data without any geometric preprocessing, but at the expense of a more demanding training and inference process. Conversely, the plate representation introduces an additional preprocessing step, yet substantially reduces the computational burden of model training and deployment.

7.4 Training

Once the dataset generation process is completed, the physics-based part of the thesis can be considered concluded, leading to the construction of `X_train`, `X_validation`, `Y_train` and `Y_validation`. At this stage, the computational part begins, and it becomes necessary to formally define the learning problem, the adopted model, and the corresponding training settings. With regard to the problem formulation, the adopted framework is naturally that of multi-label classification, implemented through a 2D Convolutional Neural Network. To define this framework, the following aspects must be introduced:

- **model architecture definition**, including the number of convolutional layers, pooling operations, activation functions, and the overall network structure;
- **parameter setting**, including the number of latent channels, learning rate, and the other hyperparameters governing the training process.

In order to define these aspects, an extensive experimental campaign was carried out, motivated by the need to explore the proposed models and the available training strategies as thoroughly as possible. As expected, shallower solutions were found to be less expressive, whereas deeper ones led to a significant increase in training time without a proportional improvement in performance. The selected investigation criterion was therefore the model performance under a fixed training-time budget. To organize this activity, the available time budget of 24 hours was divided into 8×3 intervals, each used to assess a limited set of parameter changes.
It should also be emphasized that the performance of a given implementation cannot be reliably assessed through a single scalar indicator. In fact, strong nonlinear dependencies exist among the different quantities involved in both training and validation. A simple example is provided by the classification accuracy, which is itself dependent on the decision threshold selected during the validation phase. For this reason, the assessment of each model was necessarily based on a joint interpretation of several indicators rather than on a single metric alone.
The criteria adopted to approve a model can be summarized as follows and are defined as **validation consistency**:

- a predominantly diagonal confusion matrix, corresponding to a satisfactory classification accuracy;
- the absence of physically inconsistent labels, such as combinations of modal classes that cannot occur in reality, corresponding to the actual learning capability of the model;
- sufficiently flat metric trends during the threshold evaluation phase, corresponding to a more reliable and confident prediction behavior, typically associated with relatively high threshold values (approximately greater than 0.4, consistently with the training setting).

At the end of this investigation, four different CNN implementations were developed and tested, denoted as **A**, **A+**, **B**, and **C**. These versions represent a progressive evolution of the same fully convolutional 2D framework and can be ordered according to an increasing speed-robustness trade-off: version A is the simplest baseline, whereas version C is the most robust and configurable solution.

For all of them, a **common baseline setup** can be identified. This baseline is the result of several stability analyses, which made it possible to distinguish between parameters that can be varied and parameters that must remain fixed, together with the specific design choices introduced in each code version.
During training, model performance is monitored not only in terms of loss, but also through F1-score, precision, and recall. For metric evaluation, the predicted probabilities are binarized using a fixed threshold. Depending on the considered implementation, this threshold is set to either 0.3 or 0.4, with 0.3 being used in the most stability-oriented configurations. In addition, the gradient norm is tracked in order to identify possible correlations between gradient spikes and abrupt variations in the metric trends.
Finally, all trained parameters, architectural settings, optimization states, and monitoring curves are saved in dedicated structures called `cnn_parameters`, so that the trained model can be reused during the subsequent validation and testing phases.

7.4.1 Basic Training Experiment Setup

In order to perform a meaningful comparison between the models, it is necessary to keep as many parameters as possible fixed.

Convolutional Settings The kernel sizes are set to 5×5 for the first layer, 3×3 for the intermediate layers, and 1×1 for the output layer. Variations of the kernel dimensions did not produce appreciable improvements for this application; therefore, a standard and recurrent convolutional structure was retained.
The network is composed of four hidden convolutional layers, characterized by 32, 64, 64, and 32 filters, respectively, whereas the last layer produces one output channel for each class. ReLU activation is applied after each hidden convolution. The baseline configuration also includes 8 latent channels.
All convolutional operations are performed with `same` padding, so that the spatial dimensions remain consistent throughout the network, except for the first layer, where a stride equal to $[2, 2]$ is introduced in order to perform an initial downsampling. This choice reduces the spatial resolution at an early stage and helps limit an excessively specific adaptation to the training patterns. The following layers use unit stride, unless otherwise specified in the considered implementation.
At the end of the convolutional stack, Global Average Pooling (GAP) is applied over the two spatial dimensions. In this way, the spatially distributed latent representation is condensed into a class-wise output vector for each sample. Depending on the considered implementation, the network may return either logits or sigmoid-activated probabilities, consistently with the adopted multi-label classification setting. In this context, **logits** are the raw output values produced by the network before the application of the sigmoid activation function and are therefore not bounded between 0 and 1. Their definition is close to that of a score, but the term *logit* specifically refers to the input of the final activation function.

Optimisation Settings The optimization process is performed through the *Adam* algorithm with a fixed learning rate equal to $2 \cdot 10^{-4}$. This value is commonly adopted together with *Adam* and was selected after preliminary tests. In particular, learning rates in the range $[5 \cdot 10^{-3}, 2 \cdot 10^{-4}]$ produced oscillatory behavior, whereas values lower than $2 \cdot 10^{-4}$ did not guarantee convergence and remained strongly dependent on the characteristics of the training dataset.

Weight initialization is performed according to a **He-uniform** strategy, which is consistent with the use of **ReLU** activations and is particularly suitable for image-classification-oriented convolutional architectures.
The training objective is defined through a multilabel cross-entropy loss. Since the problem is formulated as a true multi-label classification task, the network output is interpreted independently for each class.
All the tested models process an input tensor of size $[n \times n \times 3 \times N]$, where $n = 100$.

7.4.2 Training Stabilization Strategies

The main differences among versions A, A+, B, and C concern the output formulation, the downsampling strategy, and the adopted training procedure.
Among the main issues arising during the training phase, it is worth mentioning loss instability, lack of convergence, and poor generalization capability on the provided inputs. Several strategies can be adopted to mitigate these problems, although they generally increase the computational cost of each training iteration.

Weight Decay - L_2 Regularization Weight decay is widely used in deep learning for image classification tasks[48]. Its main purpose is to limit excessively large weight values, which may lead to overfitting and unstable training dynamics. In the considered implementations, weight decay was introduced as an L_2 regularization term applied to the convolutional weights, as defined in Eq.3.25.
More precisely, the standard loss function is augmented by a penalty proportional to the squared L_2 norm of the weights. This encourages smaller parameter values and improves the robustness of the optimization process.

Gradient Clipping Gradient clipping is a stabilization strategy in which the optimization step is limited whenever the gradient norm becomes excessively large[50]. In this thesis, a basic value-based gradient clipping formulation is adopted:

$$\nabla \tilde{g} = \min\{\max(\nabla g, -\gamma), \gamma\} \tag{7.21}$$

where γ is a fixed clipping threshold. Although more advanced adaptive clipping strategies exist, this simpler formulation was considered sufficient to limit excessively large updates and improve training stability.
For a fixed maximum number of iterations, gradient clipping, as well as its normalized-gradient variant, generally leads to faster and more stable convergence than standard gradient descent. In particular, it helps reduce oscillations in both the loss evolution and the training metrics.

Batch Normalization Batch normalization is a technique in which the activations of hidden layers are normalized during training. The underlying idea is that, as the parameters are updated, the distribution of intermediate activations may vary significantly, making the optimization process more difficult[51]. By keeping these activations more stable, batch normalization facilitates training, accelerates convergence, and may improve robustness. In the tested CNN implementations, Batch Normalization was included in the intermediate convolutional blocks as an additional mechanism to regularize and stabilize the learning process.

7.4.3 Comparison of the Tested CNN Architectures

Version A. Version A is the simplest and fastest baseline, although with limited control over numerical stability. In this implementation, no explicit stabilization strategies are introduced. The final output is passed through a sigmoid function to obtain multilabel probabilities directly. Downsampling is relatively aggressive, since the first three convolutional layers use stride 2 by default. This reduces the computational cost of each iteration, but also causes an early loss of spatial information. The loss function is the multilabel cross-entropy computed on probabilities. As observed during the experiments, this version is more prone to spikes in the training curves and therefore exhibits limited loss stability.

Version A+. Version A+ preserves the same overall architecture as version A, but introduces several modifications aimed at improving robustness while keeping the computational cost moderate. In particular gradient clipping is applied with threshold equal to 1.0 is introduced in order to reduce unstable parameter updates. Batch normalization becomes optional rather than mandatory, allowing the model to adapt more flexibly to the training setting. Therefore, version A+ can be regarded as a more stable refinement of the baseline model, with improved training control and reduced sensitivity to oscillatory behavior.

Version B. Version B introduces a more advanced and numerically stable training formulation. From the architectural point of view, it still follows the same fully convolutional design, but the network output is now expressed in terms of logits rather than probabilities. The sigmoid activation is applied only when computing evaluation metrics. This enables the adoption of a numerically stable binary cross-entropy with logits, implemented through a softplus-based formulation. In addition, version B supports optional mini-batch training, gradient clipping based on the global gradient norm, optional L_2 weight decay, and an alternating stride pattern of the form 1-2-1-2-1 across the five convolutional layers. Monitoring frequencies for metrics and visualization are also configurable, which improves computational efficiency during long runs. As a result, version B represents a clear improvement in robustness and flexibility with respect to versions A and A+.

Version C. Version C is the most complete and robust implementation among the tested variants. It inherits the logits-based formulation of version B, together with stable binary cross-entropy, gradient clipping by global norm, optional weight decay, and support for both full-batch and mini-batch training. In addition, Batch Normalization is fully integrated into the initialization and forward-propagation logic, thus making the model more extensible while preserving a stable default behavior. For these reasons, version **C** can be regarded as the most advanced solution from the implementation point of view.

7.4.4 Final Assessment

The four implementations correspond to successive improvements of the same core CNN model. Version A emphasizes simplicity and execution speed, but offers limited protection against unstable optimization. Version A+ improves training stability through milder downsampling and gradient clipping. Version B introduces a more rigorous logits-based loss formulation and a significantly more flexible training pipeline. Finally, version C combines these improvements into the most robust and reusable implementation.
In summary, the progression from A to C reflects the transition from a compact baseline toward a more stable, configurable, and reproducible CNN solution.

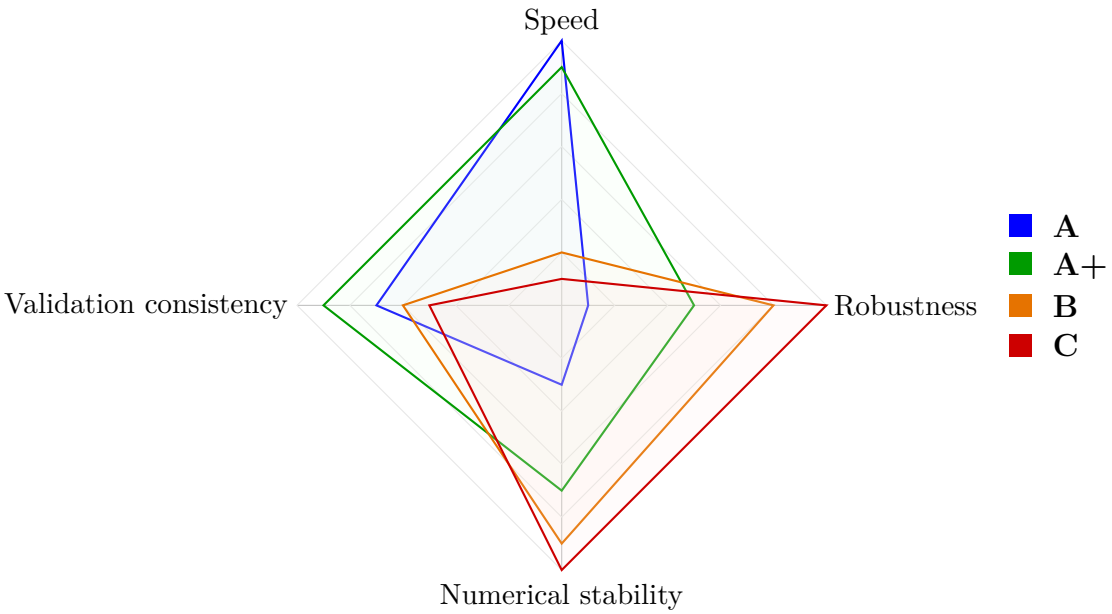


Figure 7.8: Qualitative radar-chart comparison of the four CNN implementations

Within the adopted comparison framework, the best overall performance was achieved by version A+. The main reason is that it provides a more favorable balance than version A between robustness and computational efficiency, allowing the loss to decrease rapidly while maintaining sufficiently stable training dynamics. In the considered time window, this behavior made version A+ more competitive than versions B and C, whose optimization process remained more conservative and whose loss values stayed significantly higher.
This result does not imply that versions B and C are intrinsically worse. On the contrary, they are more robust and methodologically more advanced. However, under the selected experimental constraints, their higher level of numerical protection and configurability did not translate into a sufficiently large practical advantage. In other words, for the present dataset and for the imposed training-time budget, the additional robustness of the most advanced implementations turned out to be partially unnecessary.
This observation further supports the choice of explicitly including training time among the evaluation criteria. In the presence of datasets characterized by highly variable and imbalanced features, adopting a heavier and more sophisticated code may increase the potential ceiling in terms of accuracy, but such an improvement is not always justified in practice. Moreover, because the modal patterns may exhibit substantial variability, a model that preserves fewer fine-scale details is not necessarily less effective in generalization. Conversely, highly detailed models may become more sensitive to features that are only weakly repeated in the testing set.
Starting from version A+, an additional refined configuration was therefore developed. This final variant preserves the overall philosophy of version A+, while further restricting the use of stride 2 to the first two layers only, in order to achieve a more balanced compromise between early information reduction and the ability to retain useful spatial details.
In conclusion, the final CNN configuration adopted in this thesis is the one summarized in Tab.7.5.

Parameter	Value	Description
learning_rate	$2 \cdot 10^{-4}$	Learning rate used by the ADAM optimizer.
num_max_iter	5000	Maximum number of training iterations.
gradient_clip_value	1.0	Threshold used for gradient clipping.
conv_kernel_sizes	{5×5, 3×3, 3×3, 3×3, 1×1}	Kernel sizes.
conv_num_filters	[32, 64, 64, 32]	Number of filters in the hidden convolutional layers.
num_latent_channels	8	Number of latent channels considered in the model definition.
num_conv_layers	5	Total number of convolutional layers, including the output layer.
conv_strides	first three layers: $[2, 2]$; others: $[1, 1]$	Stride adopted in the convolutional layers through the default forward configuration.
padding	same	Padding strategy adopted in all convolutional layers.
activation function	ReLU	Non-linearity applied after each hidden convolutional layer.
output activation	Sigmoid	Final activation used for multi-label probability estimation.
threshold_train	0.3	Threshold used to compute metrics during training.
weight initialization	He uniform	Initialization rule adopted for convolutional weights.
pooling	Global average pooling	Spatial pooling applied at the end of the convolutional stack.

Table 7.5: Main training and architectural parameters of the final AIseeU model.

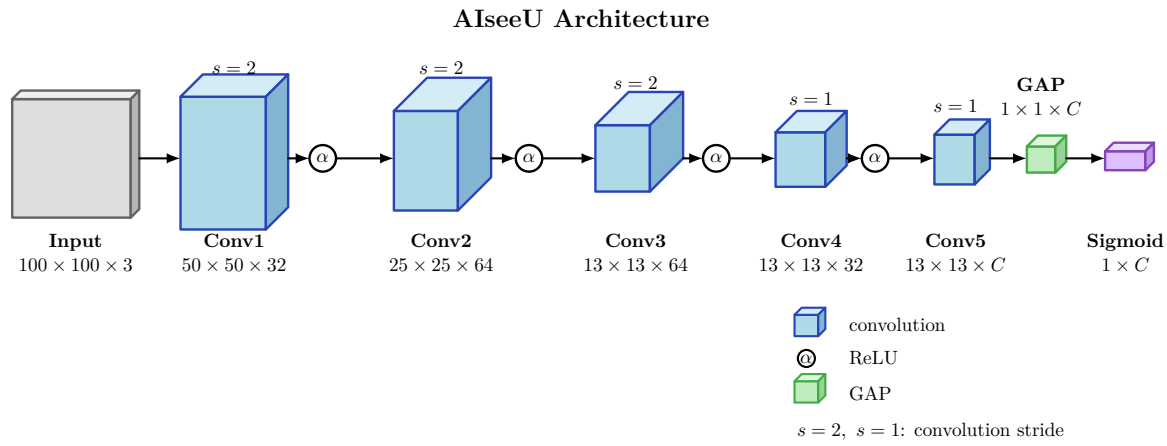


Figure 7.9: Final CNN 2D architecture.

7.4.5 Training Results

At the end of training, it is possible to inspect the loss, the norm of the gradient, and the main performance metrics. Two pronounced spikes are observed at approximately iterations 1500 and 4000. These peaks are associated with temporary gradient instabilities, which produce corresponding spikes in the loss and a sharp deterioration of the monitored metrics. Nevertheless, the overall trends remain convergent for all the considered quantities, showing that the algorithm successfully recovers after these unstable events and continues progressing toward convergence.

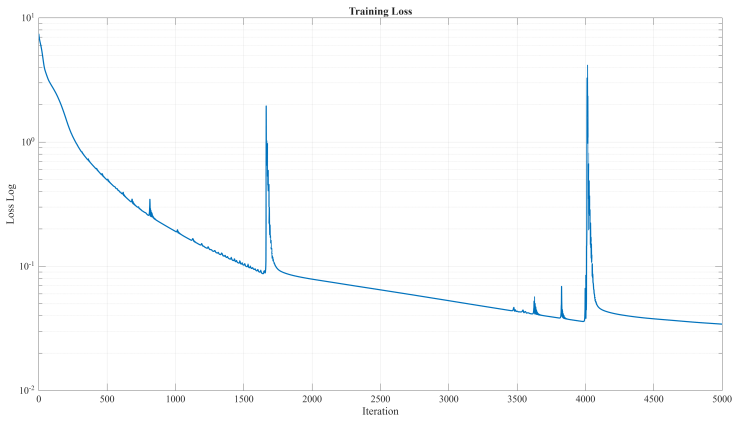


Figure 7.10: Training loss log.

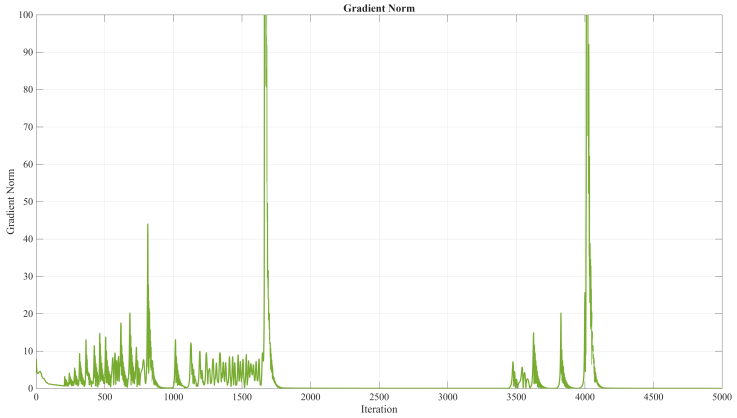


Figure 7.11: Norm of the gradient.

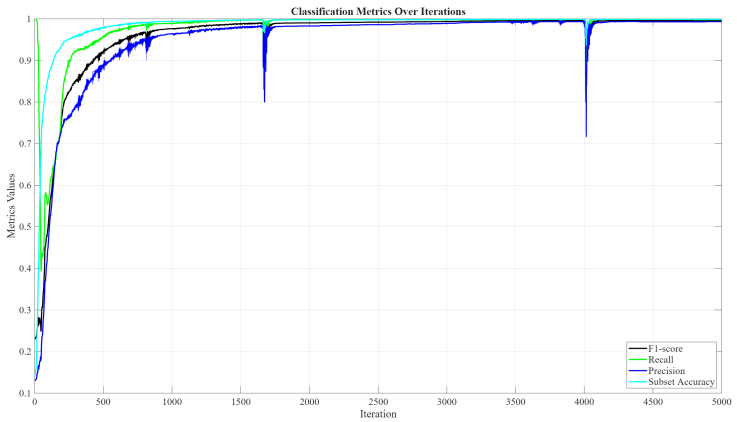


Figure 7.12: Metrics.

7.5 Validation

During validation, the confusion matrix can be analysed and the classification threshold can be selected by maximizing the $Micro - F_1$ score.
Since the obtained values are only slightly lower than those observed during training, it can be argued that no significant overfitting is present. The percentage variation in the metrics is more likely due to differences in mode shapes from one engine to another.
Moreover, the metric curves as a function of the threshold exhibit an almost flat trend, which provides evidence of the stability of the proposed approach. In addition, the fact that the optimal threshold is relatively high suggests that the network makes its predictions with a high degree of confidence, reflecting the robustness of the learned features.

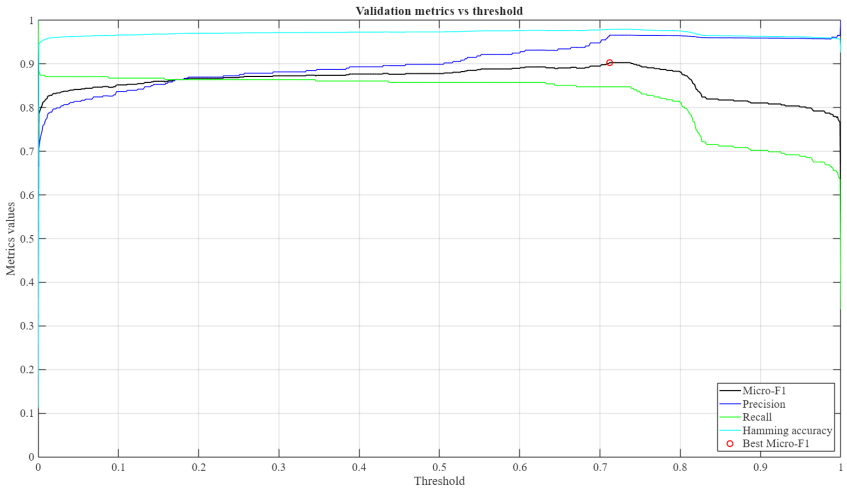


Figure 7.13: Threshold evaluation.

The confusion matrix shows very good performance for the **low-frequency modes** (EW, F, T, and TW), whereas the network appears to be less effective in generalizing the learned features to the **high-frequency modes** (2T, 2FL, and 2F). From a physical perspective, this behavior is reasonable, since high-frequency modes often appear in multiple forms that can still be recognized by the human eye as belonging to the same family, while the model seems to struggle in identifying equally robust correlations.
Additional difficulties arise from the presence of **transition modes**. In the validation set, all the 2F samples correspond to transition cases; therefore, the boundary between 2FL and 2F is not sharply defined. This issue was expected, since the training set contains modes with much clearer and more distinct boundaries. As the nodal diameter (ND) sampling becomes denser, it becomes increasingly difficult for the model to determine whether a given sample should be associated with one label or the other. This effect is clearly reflected in the observed misclassifications. In this respect, it is actually notable that the network is still able to generalize correctly for some 2FL samples.
The main limitation is represented by the 2T modes, for which the model does not seem to have identified sufficiently reliable and discriminative features.
Finally, the T+2F case is characterized by a shape that was not present in the training set, which consequently leads to a misclassification. Nevertheless, the model still appears to capture the underlying T component of the mode.
Overall, the confusion matrix can be considered physically consistent and, to a large extent, justified by the strong imbalance between the training and validation datasets.

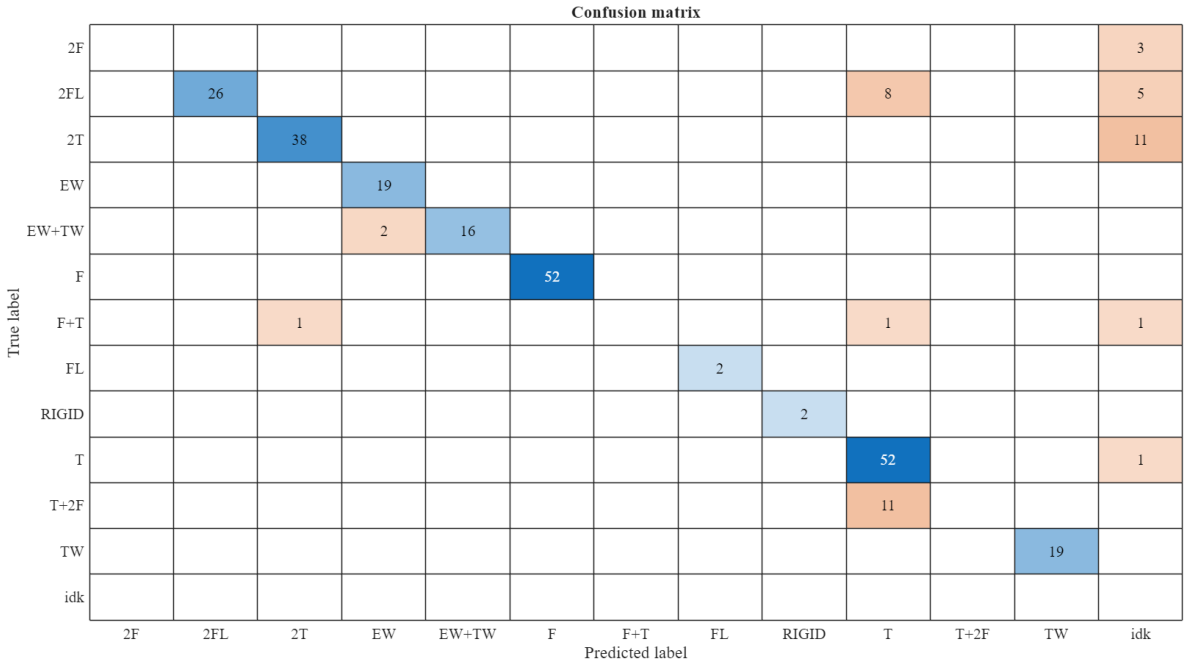


Figure 7.14: Validation confusion matrix.

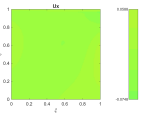
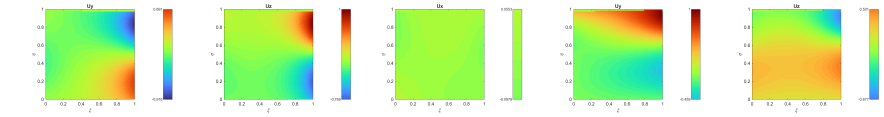
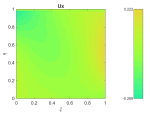
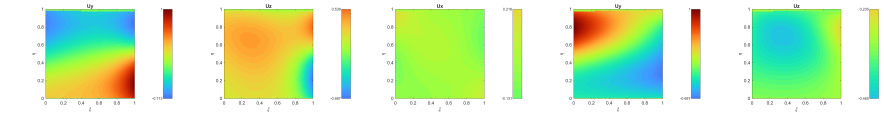
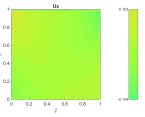
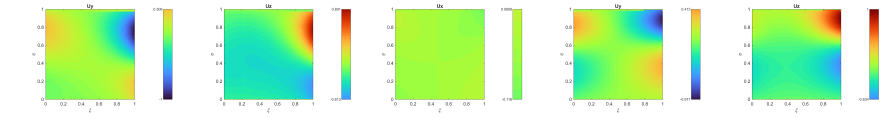
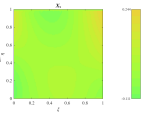
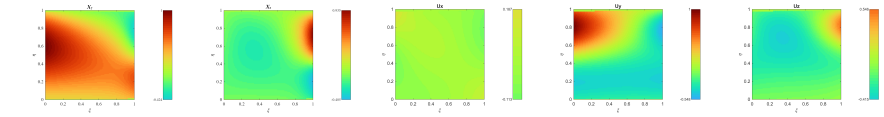
Mode	Training	Validation
2F		
2FL		
2T		
T+2F		

Table 7.6: Comparison between training and validation modes.

8 - Results

8.1 Testing

The main outcome of the project is the FreND obtained during the testing phase, shown in Fig.8.1. This testing campaign was carried out on an engine for which FreND was not previously available, thus providing a meaningful benchmark for evaluating the generalization capability of the proposed methodology.

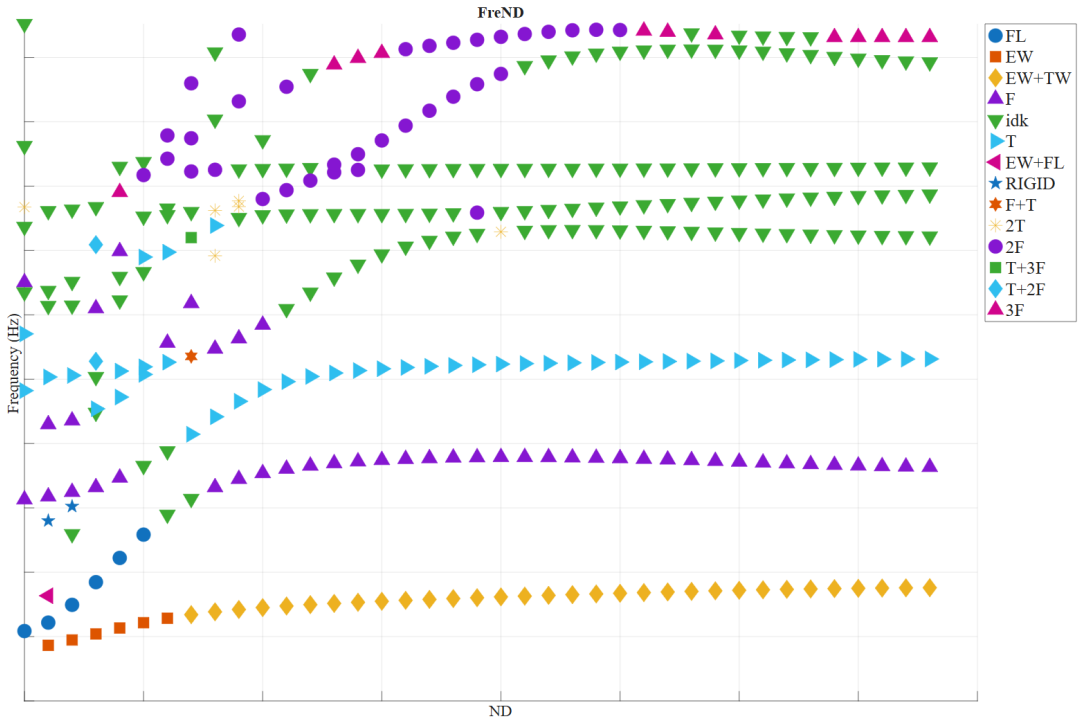


Figure 8.1: FreND obtained during the testing phase.

As highlighted in Fig.8.2, different regions of the FreND can be identified.
First, the **green region** is mainly populated by low-frequency modes, which are also the easiest to excite in practice. In particular, three main modal families were expected in this region: an EW mode gradually evolving into a TW mode; a second family composed of FL modes transitioning into F modes; and a third family which, in the true classification, consists of F modes evolving into T modes at high nodal diameters. As can be observed, in this area the proposed code performs very well. The only noticeable exception is represented by transition modes; however, even in these cases, the network often returns an idk response rather than an incorrect classification. This behavior is still desirable, since, in an engineering context, expressing uncertainty is preferable to providing a misleading prediction.

Second, the **yellow region** contains mostly transition modes. Although these modes are intrinsically more ambiguous and therefore harder to classify, the network still provides reasonably acceptable results, showing a fair degree of robustness.
Finally, the **red region** contains modes that are difficult to excite and, consequently, also difficult to identify. These correspond, respectively, to the 2F, 2T, and 3F modes. In this region, the network shows its main weaknesses, and a significant drop in performance can be observed.
Another relevant aspect is that, for this engine, the system modes - on which the neural network was not explicitly trained - do not significantly affect the recognition of the blade mode. In other words, the contributions of the disk and seal do not contaminate the modal shape strongly enough to prevent the correct identification of the blade-related mode. This is an encouraging result, as it suggests that the network is able to focus on the dominant blade features even in the presence of additional structural contributions.

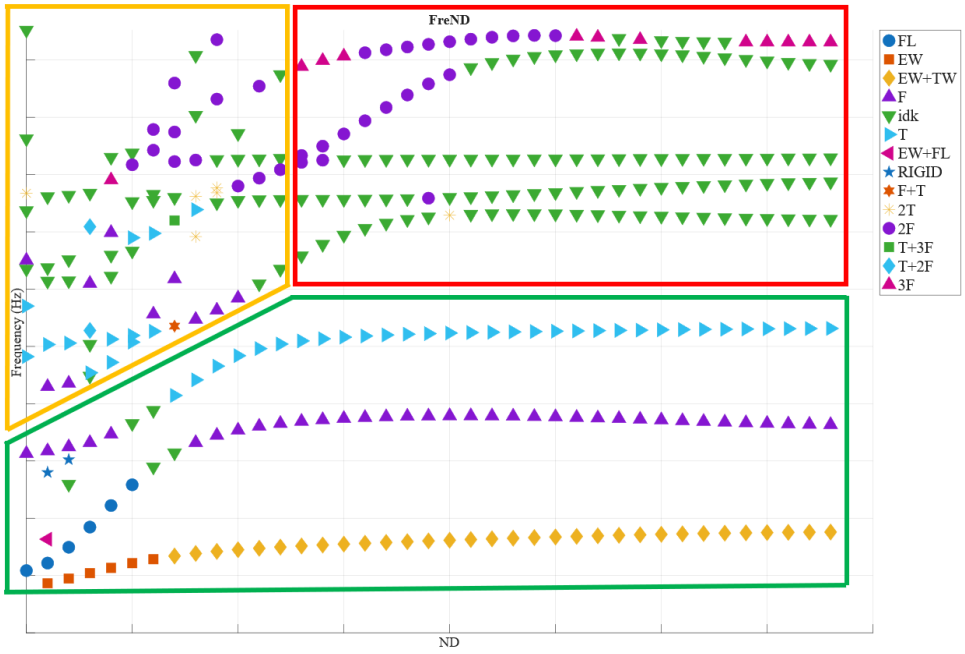


Figure 8.2: Interpretation of the main regions of the testing FreND.

8.2 Problems and Possible Improvements

From a computational point of view, the project presents several critical issues that should be addressed in future developments.

- **Strong imbalance of the training set.** The training dataset is highly unbalanced, which makes the generation of synthetic modes almost unavoidable. This is already a delicate step when very detailed geometries are considered. A possible solution would be the implementation of a macro able to automatically define the blade CAD geometry through N parameters, depending on the selected model. By varying these parameters within a properly defined design space and solving the corresponding FEM models, it would be possible to generate a much broader and more balanced database. In particular, it would be useful to focus on low nodal diameters (ND), so as to avoid the generation of too many nearly identical modes at high ND and reduce the imbalance problem.

Special attention should also be given to system modes. Such a database could later be exploited not only for classification tasks, but also for regression problems aimed at predicting the aeroelastic behavior of the blade directly from the excitation conditions.

- **High intra-class variability of modal shapes.** The same modal family may appear in several different forms, with significant variability in terms of direction, localization of the displacement maxima, and spatial pattern over the blade surface. For this reason, a more detailed labeling strategy would be beneficial, where each label corresponds to a specific sub-type of the mode. In other words, the labeling should be mathematically grounded on how the mode manifests itself: which family characterizes the blade, along which direction it develops, in which area the maximum displacement occurs, and which geometric pattern is observed. An example of this possible approach is reported in Tab.8.1. This refinement would be especially useful for the T and F families, which are characterized by high variability. Such an approach was not implemented in the present work, partly because it does not reflect the current industrial labeling practice adopted at *AvioAero*, and partly because its implementation is not trivial. Nevertheless, this strategy could reduce ambiguity during training and, at the same time, enable a more flexible post-processing stage in which sub-classes are eventually merged into broader engineering categories.

- **Need for a deeper network.** The structural basis of the problem should be reconsidered in greater detail. Due to time constraints, it was not possible to carry out a comprehensive theoretical investigation of the structural dynamics underlying the observed modal patterns. A more advanced theoretical analysis would likely help in defining better classes, improving the preprocessing pipeline, and understanding the physical limits of the classification task itself.

- **From multi-label to multi-class.** Since the current manual approach tends to assign each sample the smallest possible number of labels, the problem is more naturally framed as a multi-class classification task. Accordingly, a more appropriate interpretation would be to adopt a multi-class approach, including transition modes, rather than a multi-label framework.

- **Use of Python libraries.** Due to the high computational cost associated with more advanced neural network architectures, such as models B and C, several strategies were explored to reduce execution time. Among them, the use of GPU acceleration was considered the most natural solution, given the well-known advantages of GPUs in neural network training and inference. However, this approach led to file corruption issues during the conversion process, the origin of which has not yet been fully understood. A more robust and promising alternative would therefore be the direct implementation of the code in Python, making use of dedicated deep-learning libraries. This solution would likely provide a more efficient computational framework, improve scalability, and significantly reduce training and inference times.

- **Use of the MAC factor.** The Modal Assurance Criterion (MAC) is a widely used indicator for quantifying the similarity between modal shapes. An additional ML-based module could be implemented downstream of the image-classification stage in order to assess uncertain predictions and, when necessary, correct potentially misclassified cases.

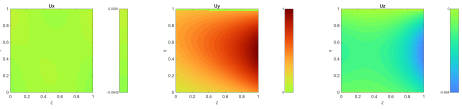
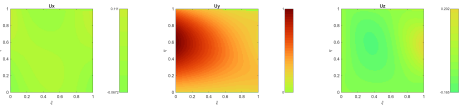
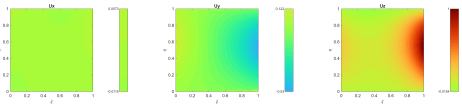
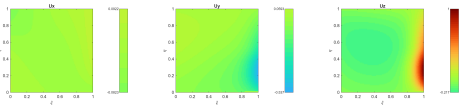
Label	Maps	Description
FYTE		F-type mode in the Y direction, mainly involving the trailing edge line (TEs).
FYLE		F-type mode in the Y direction, mainly involving the leading edge line (LEs).
F1TE		F-type mode primarily concentrated on the trailing edge line.
F2TE		F-type mode involving the trailing edge line, with possible transition toward more complicated modes.

Table 8.1: Example of a possible refined labeling strategy for F-type modes.

8.3 Conclusions

This project led to the development of `AIseeU`, an artificial-intelligence-based tool designed for the labelling of modal shapes for aeroelastic applications.
A crucial part of the work was the data-processing stage, which represented the main physical foundation of the project. In this phase, the frames corresponding to the maximum real displacement were isolated, as these were found to be sufficient for modal recognition. Alternative strategies were also considered and preliminarily investigated; however, this aspect remains an open question and deserves further study.
The computational implementation was carried out in commercial softwares and resulted in the development of a set of codes capable of performing image-based recognition independently of the specific input source. This confirms the flexibility of the proposed workflow and its potential applicability to similar problems.
Despite the limited initial experience in the field, the project achieved a significant result: the developed tool performs reliably on the most critical modes, namely the low-frequency modes, which are also the most relevant from an aeroelastic point of view because they are more easily excitable. This is undoubtedly one of the main strengths of the work.
On the other hand, the tool still shows difficulties when dealing with more complex modes, such as 2F and 2T. The analysis carried out in this thesis suggests that these limitations are mainly due to the distribution of the available inputs, which is not well suited to the training requirements of neural networks. In this sense, the problem is not only algorithmic, but also structural and data-driven. For this reason, several practical directions for future improvements have been proposed, including a more balanced dataset, a more detailed labeling strategy, and a deeper physical interpretation of the modal families.
In conclusion, the work can be considered successful in achieving its main objective: demonstrating the feasibility of using a 2D CNN for the recognition of blade modal shapes in an aeroelastic framework. At the same time, it has highlighted the main limitations of the current approach and laid the groundwork for future developments toward a more robust, physically informed, and industrially applicable tool.

"Eccetera ... perché la minestra si fredda"
— Leonardo da Vinci

9 - Appendix A

9.1 Turbo-engines analysed

This thesis focuses on the main civil products of *GE Avio Aero*.[1]

9.1.1 GE9X

The General Electric GE9X is a high-bypass-ratio turbofan developed exclusively for the Boeing 777X. In 2019, it recorded the highest thrust ever achieved by a commercial aircraft engine and, upon its entry into service in 2020, it became the largest and most powerful commercial aircraft engine ever produced. In addition, the GE9X is one of the most fuel-efficient and quietest engines in its class. This model is also characterized by composite fan blades and the adoption of ceramic matrix composites (CMCs), which contribute to improved thermal performance.

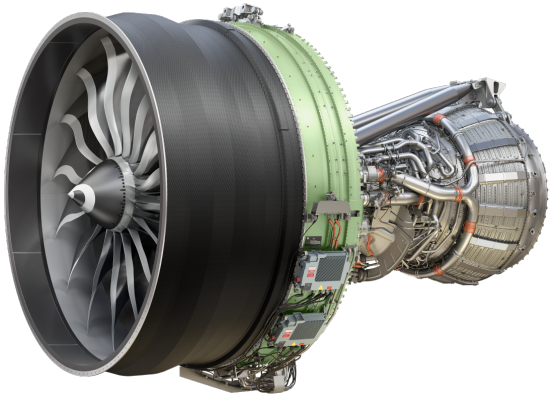


Figure 9.1: GE9X.

Parameter	**Value**
Engine type	Turbofan
Maximum thrust	$\sim$ 597 kN
Fan diameter	$\sim$ 3.40 m
Overall pressure ratio	$\sim$ 60 : 1
Bypass ratio	$\sim$ 10 : 1
Entry into service	2020

9.1.2 GE90

The General Electric GE90 is a turbofan engine developed and produced by GE Aviation since the 1990s for the Boeing 777. In its -115B variant, it is one of the most powerful aircraft engines ever built and is surpassed in size only by its successor, the GE9X. Among its most significant technological innovations are the introduction of composite fan blades, bleed valve doors designed to withstand foreign object debris (FOD), and the first FAA-approved additive-manufactured compressor sensor. The GE90 also achieved an exceptionally high reliability rate of 99.98%.

[1]The values and images reported in this appendix are publicly accessible through the official GE Aerospace website: https://www.geaerospace.com/commercial and https://www.gevernova.com/gas-power/products/gas-turbines, except for GE90 https://www.mtu.de/fileadmin/_processed_/6/9/csm_maintenance_civil_aircraft_engine_portfolio_ge90_b7d5854fba.jpg.

Figure 9.2: GE90.

Parameter	Value[1]
Engine type	Turbofan
Maximum thrust	$\sim$ 511 kN
Fan diameter	$\sim$ 3.43 m
Overall pressure ratio	$\sim$ 42 : 1
Bypass ratio	$\sim$ 9 : 1
Entry into service	1995

[1] GE90-115B variant.

9.1.3 LMS100

The LMS100 PA is one of the largest and most efficient aeroderivative gas turbines currently available, and it is capable of reaching full power in less than 10 minutes. The LMS100 consists of a low-pressure compressor, an intercooler, a supercore, and a power turbine. This architecture allows the machine to combine high efficiency with operational flexibility, making it particularly suitable for power generation applications.

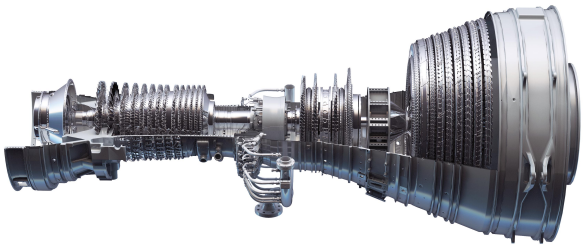


Figure 9.3: LMS100.

Parameter	Value[2]
Engine type	Gas turbine
Power output	$\sim$ 110 MW
Thermal efficiency	$\sim$ 52% LHV
Entry into service	2006

[2] Simple cycle.

9.1.4 RISE

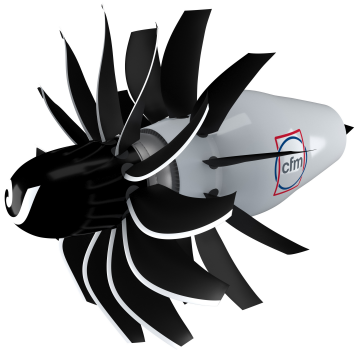


Figure 9.4: RISE.

The CFM RISE (Revolutionary Innovation for Sustainable Engines) program is focused on the development of next-generation propulsion technologies aimed at significantly improving efficiency and sustainability. Its key innovations include an open-fan architecture, designed to provide a step change in fuel efficiency with particular attention to durability, and the use of supercomputing techniques to optimize the design and reduce noise levels with respect to current engines.
In addition, RISE explores integrated hybrid-electric propulsion systems, with the objective of demonstrating a megawatt-class hybrid powertrain. Finally, the technology is being developed for compatibility with unblended sustainable aviation fuel (SAF), while also supporting future hydrogen-fuel capability.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016. https://www.deeplearningbook.org

[2] D. P. Kroese, Z. I. Botev, T. Taimre, S. Vaisman, and R. Salomone, *Solutions Manual to Accompany Data Science and Machine Learning: Mathematical and Statistical Methods*, Jan. 8, 2020.

[3] V. Latorre, *Neural Networks, Surrogate Models and Black Box Algorithms: Theory and Applications*, Ph.D. thesis, Università La Sapienza di Roma, Dottorato in Ricerca Operativa, XV ciclo, Academic Year 2011–2012.

[4] E. A. Bender, *Mathematical Methods in Artificial Intelligence*, IEEE Computer Society Press, Los Alamitos, CA, 1996. ISBN: 0-8186-7200-5.

[5] MathWorks, *Machine Learning*, https://www.mathworks.com/discovery/machine-learning.html

[6] H. Younes, M. Alameh, A. Ibrahim, M. Rizk, and M. Valle, "Efficient Algorithms for Embedded Tactile Data Processing," pp. 113–138, River Publishers, 2020.

[7] MathWorks, *Overfitting*, https://www.mathworks.com/discovery/overfitting.html

[8] N. Spolaôr, E. A. Cherman, M. C. Monard, and H. D. Lee, "A Comparison of Multi-label Feature Selection Methods Using the Problem Transformation Approach," *Electronic Notes in Theoretical Computer Science*, vol. 292, pp. 135–151, 2013. https://doi.org/10.1016/j.entcs.2013.02.010

[9] O. O. Awe, *Computational Strategies for Tackling Imbalanced Data in Machine Learning*, IASC Webinar, 2024.

[10] Notes by Lex, *SMOTE*, https://notesbylex.com/smote

[11] MathWorks, *ROC Curve*, https://www.mathworks.com/discovery/roc-curve.html

[12] Analytics Vidhya / Medium, *Undersampling and Oversampling: An Old and a New Approach*, https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392

[13] P. Achlioptas, *Stochastic Gradient Descent in Theory and Practice*, Stanford University.

[14] R. Tibshirani, *Lecture 5: Gradient Descent*, Carnegie Mellon University, lecture notes.

[15] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[16] S. S. Du, *Gradient Descent for Non-convex Problems in Modern Machine Learning*, CMU-ML-19-102, Machine Learning Department, Carnegie Mellon University, 2019.

[17] G. Farina, *Lecture 15: Stochastic Gradient Descent*, Massachusetts Institute of Technology, 2025.

[18] University of Warwick, *Machine Learning Reading Group Presentation*, https://warwick.ac.uk/fac/sci/mathsys/news/readinggroups/machinelearningrg/presentation.pdf

[19] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, "A Survey and Taxonomy of Loss Functions in Machine Learning," *AI*, vol. 7, no. 4, art. 128, 2026. https://doi.org/10.3390/ai7040128

[20] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[21] C. E. Shannon, *A Mathematical Theory of Communication*, *Bell System Technical Journal*, vol. 27, pp. 379-423 and 623-656, 1948.

[22] PyTorch Documentation, *MultiLabelMarginLoss*, https://docs.pytorch.org/docs/2.12/generated/torch.nn.MultiLabelMarginLoss.html

[23] A. Tomar and P. Laxkar, "Differences of Tanh, Sigmoid and ReLU Activation Function in Neural Network," *International Journal of Scientific Progress and Research (IJSPR)*, vol. 80, no. 6, June 2022

[24] Z. C. Lipton, C. Elkan, and B. Naryanaswamy, *Thresholding Classifiers to Maximize F1 Score*, University of California, San Diego, 2014.

[25] J. Domke, *Empirical Risk Minimization and Optimization*, University of Massachusetts Amherst, course notes.

[26] T. Szandała, *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, Wroclaw University of Science and Technology, Wroclaw, Poland.

[27] R. Tibshirani, *Stochastic Gradient Descent*, Convex Optimization 10-725, Carnegie Mellon University, lecture notes.

[28] S. Jelassi and Y. Li, *Towards Understanding How Momentum Improves Generalization in Deep Learning*, 2022.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[30] Stanford CS230, *Section 4*, https://cs230.stanford.edu/section/4/

[31] PyTorch Documentation, *torch.nn.init*, https://docs.pytorch.org/docs/2.12/nn.init.html

[32] APXML, *Initialization Strategies: Xavier and He*, https://apxml.com/courses/deep-learning-regularization-optimization/chapter-7-optimization-refinements-tuning/initialization-strategies-xavier-he

[33] D. H. Wolpert and W. G. Macready, *No Free Lunch Theorems for Optimization*, *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, 1997.

[34] W.-Y. Loh, *Classification and Regression Trees*, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, Jan. 2011. https://doi.org/10.1002/widm.8

[35] MathWorks, *Support Vector Machine*, https://www.mathworks.com/discovery/support-vector-machine.html

[36] J. L. Rojo-Álvarez, M. Martínez-Ramón, J. Muñoz-Marí, and G. Camps-Valls, "Support Vector Machine and Kernel Classification Algorithms," in *Digital Signal Processing with Kernel Methods*, 1st ed., Wiley-IEEE Press, 2018, pp. 433–502. https://doi.org/10.1002/9781118705810.ch10

[37] Cornell University, *k-Nearest Neighbors Lecture Notes*, https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

[38] Stanford Vision Lab, *CS231n kNN Demo*, http://vision.stanford.edu/teaching/cs231n-demos/knn/

[39] MathWorks, *Classification Using Nearest Neighbors*, https://www.mathworks.com/help/stats/classification-using-nearest-neighbors.html

[40] GeeksforGeeks, *Weighted k-NN*, https://www.geeksforgeeks.org/machine-learning/weighted-k-nn/

[41] MathWorks, *Classification Ensembles*, https://www.mathworks.com/help/stats/classreg.learning.classif.classificationensemble.html

[42] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, *Graph Neural Networks: A Review of Methods and Applications*, *AI Open*, vol. 1, pp. 57-81, 2020.

[43] Distill, *A Gentle Introduction to Graph Neural Networks*, https://distill.pub/2021/gnn-intro/

[44] K.-L. Du, C.-S. Leung, W. H. Mow, and M. N. S. Swamy, *Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era*, *Mathematics*, vol. 10, no. 24, art. 4730, 2022. https://doi.org/10.3390/math10244730

[45] T. E. Smith, *A Modal Aeroelastic Analysis Scheme for Turbomachinery Blading*, Final Report, Sverdrup Technology, Inc., Lewis Research Center Group, Brook Park, Ohio, prepared for NASA Lewis Research Center under Contract NAS3-25266, NASA-CR-187089, March 1991.

[46] J. R. Wright and J. E. Cooper, *Introduction to Aircraft Aeroelasticity and Loads*, 2nd ed., Wiley.

[47] A. Bhaduri, J. Li, S. Ghosh, and L. Wang, "Efficient Surrogate Modeling for Turbine Blade Row Cyclic Symmetric Mode Shapes," in *Proceedings of ASME Turbo Expo 2022: Turbomachinery Technical Conference and Exposition*, Rotterdam, The Netherlands, June 13-17, 2022, Paper No. GT2022-83414.

[48] T. N. Kipf and M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, in *International Conference on Learning Representations (ICLR)*, 2017.

[49] F. D'Angelo, M. Andriushchenko, A. Varre, and N. Flammarion, *Why Do We Need Weight Decay in Modern Deep Learning?*, Theory of Machine Learning Lab, EPFL, Lausanne, Switzerland, 2024.

[50] J. Zhang, T. He, S. Sra, and A. Jadbabaie, *Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity*, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

[51] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448-456.

[52] S. Zucca, *Slides for the Rotor Dynamics Course*, Politecnico di Torino, 2024.

[53] S. Pieraccini, *Slides for the Course Numerical Methods and Scientific Computing*, Politecnico di Torino, 2023.