UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell’Informazione e della Produzione

Corso di Laurea Magistrale in Ingegneria Informatica

Classe n. LM-32 – Ingegneria Informatica

**DEEP LEARNING PREDICTIONS OF AERODYNAMIC COEFFICIENTS: A CASE STUDY**

Relatore:

Chiar.mo Prof. Mario Verdicchio

Tesi di Laurea Magistrale di:

Andrea PAPA

Matricola n. 1076272

ANNO ACCADEMICO 2022 / 2023

**Index**

[Introduction 5](#_Toc136009266)

[1. Basic Theorical Introduction to Neural Networks and FNNs 8](#_Toc136009267)

[1.1. Neural Networks as Simulation of Nervous Systems 8](#_Toc136009268)

[1.2. Structure and Functioning of Feedforward Neural Networks 9](#_Toc136009269)

[1.3. Multilayer Perceptrons and Deep Neural Networks 10](#_Toc136009270)

[1.4. Activation Function Selection 11](#_Toc136009271)

[1.5. Optimization Algorithm and Loss Function 11](#_Toc136009272)

[1.6. Overfitting 12](#_Toc136009273)

[1.6.1. Cross Validation 12](#_Toc136009274)

[1.6.2. Regularization Methods 14](#_Toc136009275)

[2. Principles of Aerodynamical Properties of Objects 15](#_Toc136009276)

[2.1. Lift and Drag Coefficients 15](#_Toc136009277)

[2.2. Airfoils and NACA Airfoils 17](#_Toc136009278)

[2.3. Angle of Attack and Stall Angles 19](#_Toc136009279)

[2.4. Lift-to-Drag Ratio 20](#_Toc136009280)

[3. Basics of Computational Fluid Dynamics 22](#_Toc136009281)

[3.1. Definition of CFD 22](#_Toc136009282)

[3.2. CFD Preprocessing 22](#_Toc136009283)

[3.3. Simulation and CFD Postprocessing 24](#_Toc136009284)

[4. Implementation Process 26](#_Toc136009285)

[4.1. Preparatory work 26](#_Toc136009286)

[4.1.1. CFD Simulations 26](#_Toc136009287)

[4.1.2. Extraction and Transformation Operations of the Simulated Data 29](#_Toc136009288)

[4.2. Programming of the Deep Neural Network 31](#_Toc136009289)

[4.2.1. Imports 31](#_Toc136009290)

[4.2.2. Main Function call and Features Extraction 32](#_Toc136009291)

[4.2.3. Training/Validation Dataset Creation 32](#_Toc136009292)

[4.2.4. Architecture Definition 33](#_Toc136009293)

[4.2.5. Training Phase Set Up 34](#_Toc136009294)

[4.2.6. Testing Phase 34](#_Toc136009295)

[4.3. Max L/D Ratio Search 36](#_Toc136009296)

[5. Discussion of the Obtained Results and Conclusions 39](#_Toc136009297)

[5.1. Results 39](#_Toc136009298)

[5.1.1. datasets\_preparation.py 39](#_Toc136009299)

[5.1.2. deep\_neural\_network.py 39](#_Toc136009300)

[5.1.3. test\_models.py 39](#_Toc136009301)

[5.1.4. max\_lift\_drag\_ratio.py 43](#_Toc136009302)

[5.2. Analysis of the Data Achieved and Comments 45](#_Toc136009303)

[6. Product Backlog and Possible further Developments 49](#_Toc136009304)

[7. Acknowledgments 51](#_Toc136009305)

[8. Bibliography 52](#_Toc136009306)

# Introduction

This academic research project has the goal of implementing a neural network model which would represent a light and reliable tool to obtain the coefficients of the lift and drag forces acting on two-dimensional wing profiles (airfoils) at wind flows with different directionalities. Such predictions should represent a less time- and cost-consuming alternative to real world measurements and virtual wind tunnel simulations, commonly used today to acquire these coefficients.

A secondary objective of the project is to find the airfoil geometry, and the inclination of it relative to the wind direction, which maximizes the lift-to-drag ratio, a common measurement of the efficiency of the airfoils.

The implemented new system must be engineered following the principles of the incremental development model with the aim to be programmer friendly and facilitate maintenance operations and allow future improvements.

The scope of this research project was selected envisioning the potential of such instrument in the context present in the period prior to the publishment of this paper. Potential applications of such program, used as module of more complex software products, range from real-time ﬂight control of aircrafts to inverse design tool of innovative and more efficient airfoil geometries.

The underlying criticalities present in the current context that led to the development of this project and the reasons that justify the creation of such system in the aforementioned proposed form can be summarized as follows:

* Acquisition of lift and drag coefficients of historical but well-known airfoil types like the 4-digits NACA airfoils is challenging since real life calculations made by the former American federal agency NACA in the 1920s and 30s are limited and approximated. Hence, to obtain reliable data in absence of wind tunnels, professionals tend to replicate the behavior of the airfoils in a given environment designing computer fluid dynamics (CFD) model. This solution guarantees accurate results but is also very time expensive.
* There are no precise and easily augmentable software solutions that allow a fast acquisition of the lift and drag coefficients. As an instance, XFOIL, a free and open source software program commonly used to acquire the properties of the airfoils in different conditions, makes enough error on its calculations that, in the professional fields, the results generated need to be validated using wind tunnel simulation even in situations in which wind turbulence is limited [1]. In addition to this, XFOIL, developed at MIT in the 1980s, is increasingly difficult to customize requiring the knowledge of Fortran. A translated open source version of XFOIL, XFLR5 (in C++) is being phased out in favor of the recently released Flow5, which is however closed source [2].
* Currently there is no instrument that allows a direct comparison of the lift-to-drag ratios of several types of airfoils. Indeed, with each software solution previously mentioned, lift and drag data need to be gathered for each airfoil design separately and only later they can be confronted.
* Due to the well-known difficulties of traditional programming solutions in the direct calculation of turbulence effects on objects [3], deep learning architectures may result as a more efficient IT tools to study airfoils-air interactions in environments in which turbulence plays a significant role. In addition, considering possible future developments of this project, deep learning networks are a promising solution to replicate and effectively predict the complex behaviors of the very high turbulent environment rising when airfoil geometries adopt high inclination in relation to the wind direction [4].

Agile practices were adopted in order to efficiently organize the development of the various elements of the project according with the principles of the incremental development model. More precisely, a one-person adaptation of the Scrum project management framework was adopted in the course of the implementation of the system [5].

The software files related to this paper follow programming best practices. Among other principles, the code was implemented having self-explanatory variable names, adopting modularity in the coding process, and containing introductions and annotations for each block of code. These principles were adopted with the purpose to make code reading more comprehension to the general public and allow an easy process of maintenance and future development.

In all the phases of the project several software programs have seen adoption:

* ANSYS Fluent was used to perform CFD simulations needed to allow the deep neural network to be supplied by a reasonable pool of data with which implement the process of training.
* Microsoft Excel was used to gather the data obtained by the CFD simulations and compare them with the experimental data available.
* Visual Studio Code was used for the implementation of the python files in charge of the ETL operations on the training data, the definition of the deep neural network structure and of the hyperparameters necessary for its training phase, and the testing of the generated models.

In compliance with a scientific method approach, the results obtained throughout the whole project were validated by confronting them with real world experimental data whenever these were available.

The main sources used for the validation of the lift and drag coefficients, as well as their ratio, are books and official documents reporting real-life measurements performed at the Langley Research Center, a research center in Virginia (US) at the time belonging to the now-defunct National Advisory Committee for Aeronautics (NACA) and now part of the National Aeronautics and Space Administration (NASA) [6]. This, like every other source cited in this paper, was verified as trustworthy by other well-known scientific entities.

The approximations and estimated errors on the calculations are always cited and lie within a scientifically accepted range.

Every passage of the process of implementation of this project was reported and explained using information obtained through scientific papers, books and other documents edited by experts in the respective field of study.

# Basic Theorical Introduction to Neural Networks and FNNs

## Neural Networks as Simulation of Nervous Systems

The cognitive capabilities and skills of humankind as well as the ones of a large variety of animals are a result of electrical pulses originating in our brain and that propagate to other areas of it and through the rest of our body thanks to the nervous system.

The elementary units which allow the exchange of this information are the neurons. These are a special type of cells that activate when struck by electrical signals that are then routed by the neurons toward other neurons utilizing extensions of the cells called axons.

Ultimately, a sequence of signals exchanged among the various neurons will activate the area of interest, permitting the functioning of our body and mind [7].

*Neural networks* are a type of computer system developed through inspiration given by the biological world, or, more precisely, by the functioning of animals’ and people’s nervous systems.

The basic elements which constitute the architecture of these networks are multiple fully connected units called *artificial neurons* [8]. As it’s observable in *Figure 1.1*, given a set of inputs, each of these objects host a mathematical function, known in the field as *activation function,* which defines the output it passes to other connected neurons [9].

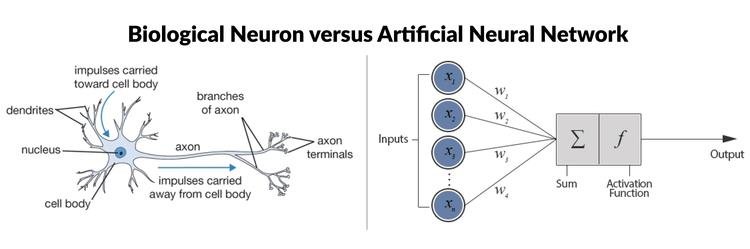


Figure 1.1.1 - Comparison of biological and artificial neurons’ structure [10]

In a process similar to the one which occurs in the natural world, neural networks use several artificial neurons in order to accomplish sets of tasks gatherable in categories like classification, regression, or anomaly detection problems. Neural networks are able to approximate the target function of a given problem through experience developed by a process of *machine learning*, a class of processes that simulate our natural process of learning. Neural networks models are generated by software files which contain information about the model structure, the learning algorithm to apply and the datasets necessary for the learning process [11]. These software files use specialized libraries known as *neural network frameworks.*

In recent years several types of neural networks, most notably deep neural networks, have demonstrated to the scientific community a great potential to solve complex problems which were hardly solvable through traditional programming solutions [9].

## Structure and Functioning of Feedforward Neural Networks

*Feedforward neural networks* (FNN) are a specific branch of neural networks characterized by the unidirectionality of the information passing through the artificial neurons which compose them due to the absence of cycles in their architecture.

The neurons constituting a FNN are divided in interconnected groups known as *layers*. The configuration in which these layers are linked with each other depends on the specific type of FNN considered. In *fully connected FNNs* each artificial neuron of a given layer has connections with all the neurons belonging to the layers preceding and following it, whenever these are present. These connections are used by the neurons to receive the information coming from the previous layer and, once their activation function was applied on the data, to route the results to all the neurons in the layer downstream. The information sent by a neuron will influence the input value of the receiving neurons according to the weight of the connection between them.

The first layer of a FNN is known as *input layer* and houses the input parameters of the neural network while the last layer is called *output layer* and hosts an array of artificial neurons whose outputs correspond to the FNN’s outputs. Depending on the type of FNN, between these two layers there could be present several artificial neurons arranged in one or more additional layers called *hidden layers*. The artificial neurons belonging to a hidden layer are also known as *hidden units* [9].

FNNs can improve iteratively the accuracy and precision of their solutions through the adoption of different *learning algorithms*. These are series of mathematical rules that enable neural networks to learn from *training datasets*, collections of data described through a series of problem characterizing features. A learning algorithm commonly adopted on FNNs is *backpropagation*. Backpropagation is a *supervised learning* technique, which means that it extracts a prediction of the output from the neural network model given the input data and it confronts it with the expected output. Calculated the loss through a selected loss function, backpropagation’s algorithm utilizes the chain rule to compute the gradient of the loss function which is then used in combination with gradient methods to adjust the weights of the connections among the neurons and lower the loss at the next iteration. These changes are made propagating the error backward through the network, giving the name to this learning algorithm [11].

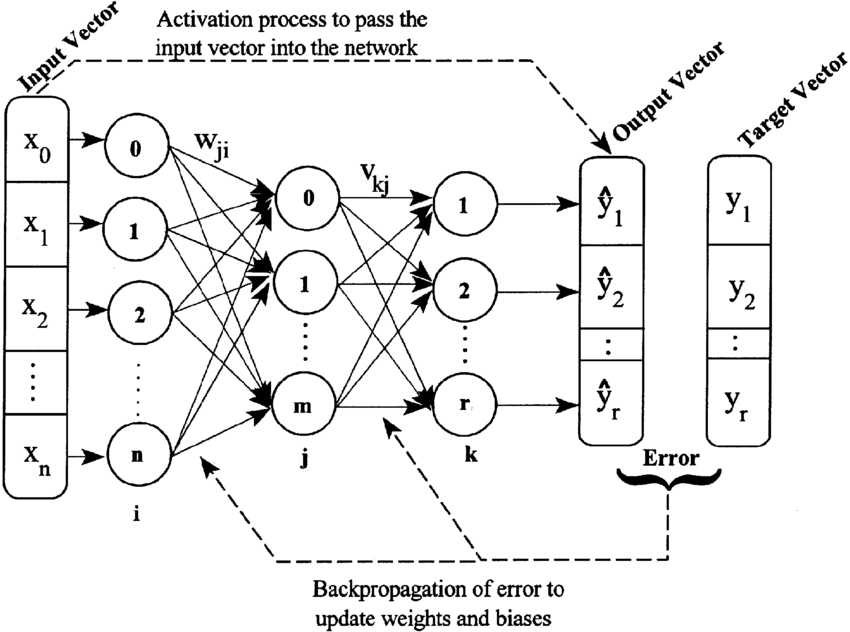


Figure 1.2 - Schematic representation of a three-layered FNN with backpropagation [12]

## Multilayer Perceptrons and Deep Neural Networks

The fully connected class of FNN in which there is at least one hidden layer is called *multilayer perceptron* (MLP).

The main advantage of MLPs, compared to the simpler single layer perceptron architectures, is their ability to distinguish and better approximate target functions which are not linear in nature, generating more precise predictions for complex problems. In particular, the higher is the number of hidden layers and hidden units in a MLP, the more degrees of freedom will the MLP have to generate a complex solution algorithm, giving it the ability to identify possible hidden information in the input data and thus better recognize patterns [13].

Due to their multilayered nature, MLPs are regarded as a basic example of deep neural network. Some newly theorized models of deep neural networks use MLP, as well as other types of FNNs, as building blocks for more complex types of deep neural networks.

A remarkable advantage of using architectures composed by interconnected FNNs over generating predictions from independent basic FNNs is the fact that this type of networks, whether they are properly structured and multiple outputs are involved, can identify hidden correlations among the outputs with high effectiveness and efficiency [14]. This enables them to solve more complex problems or at least offer more precise predictions than simpler independent machines.

In recent years, types of neural networks with multiple interconnected FNNs were used in a variety of problems such as image classification, object recognition, natural language processing, speech recognition, predictive modeling, and anomaly detection. As an instance, ChatGPT, an artificial intelligence chatbot which in the months prior to writing this paper was repeatedly on the news, is based on an architecture which includes complex deep FNNs [15].

## Activation Function Selection

There are several types of activation functions used in the artificial neurons of neural networks. These range from very simple mathematical functions to more complex ones. The choice of which activation function to adopt for a neuron depends on a different series of factors. Among others, these include the specific task of the neural network model and position of the neuron’s layer in the structure.

The most basic activation function is *identity*. The neurons with identity as activation function return outputs equivalent to the input they receive. Today it is almost exclusively used with neurons belonging to output layers of regression models.

A more complex but very popular choice of activation function for hidden units is the *leaky rectifier linear unit* (Leaky ReLU). Neurons with this activation function smooth out negative inputs while leaving unchanged the positive ones. The incidence of this process of negative values reduction depends on a predetermined “alpha” value comprised between 0 (which returns 0 for every negative value and thus is equivalent to a more traditional ReLU activation function) to 1 (which makes the neuron return an output equivalent to the input in any circumstance, like in the case of using an identity activation function). This function is preferable to ReLU since, differently from this, it is not subject to the “Dying ReLU problem”. This is a condition in which neurons can be pushed into states in which they become de-facto inactive for any possible input, reducing the model capacity [16]. The identity, ReLU, Leaky ReLU activation functions are represented in *Figure 1.3*.

Immagine che contiene antenna

Descrizione generata automaticamente

Figure 1.3 – Identity, ReLU and Leaky ReLU activation functions

## Optimization Algorithm and Loss Function

An important step in neural networks’ creation is the selection of the *optimization algorithm* to adopt. These are iterative mathematical algorithms which after each iteration edit the parameters in order to achieve the minimum (or the maximum) of a target function [17]. In the context of machine learning with supervised learning, optimization algorithms are used during the training phase of the model to find the function which minimizes the error on the training data, known in AI field as *loss*, and thus reflecting the best the input function. Depending on the specific problem on which the model is trained, adopting a certain optimization algorithm of the neural network to obtain the best prediction results.

The *Adaptive Moment Estimation* (ADAM) is one of the most commonly used optimization algorithms due to several characteristics it has. Of these the most peculiar is its use of adaptive learning rates for each of the parameters, which allows the model to converge in a short number of training iterations, known in the AI field as *epochs* [18].

Similarly, there are a variety of different loss functions. Also in this case, the choice of which one to adopt in the neural networks lies on the type of problem to solve as well as other factors.

One of the most common loss functions used in the field is *mean squared error* (MSE). This measures the average squared difference between the predictions made by the neural network and the output control values supplied to the model through the training set [19].

## Overfitting

A common phenomenon that occurs when a neural network is overly trained or overly complex is *overfitting*. In such cases the learned solution algorithm of the neural network reflects very faithfully the data it trained on, lacking, however, the ability of generalization. This leads to strong mismatches between the expected and the predicted values on data absent in the training dataset. On the other hand, an excessively simple architecture or a scarcely trained model can’t reflect the complexity of the problem and is thus overly generalized, leading to high errors. This situation is called *underfitting*.

Examples of underfitted and overfitted models are displayed in *Figure 1.4*.



Figure 1.4 - Underfitting and overfitting in a regression problem [20]

### Cross Validation

To recognize situations of overfitting (or underfitting) it is necessary to calculate the error made by the model while making predictions on data it didn’t train on and compare it with the loss on the training data. This can be achieved using the *cross-validation* technique [21]. This involves retaining some of the experimental data available and assigning them to a *validation dataset* instead of to the training dataset. During each epoch of the training phase, the data in the validation dataset are then used as input of the neural network model and, as happens with the data in the training dataset, the generated outputs are then confronted with the expected ones. Terminated the training phase, the losses obtained on the two datasets can then be confronted to evaluate whether overfitting has occurred. In case the difference between these two values is high the model could be overfit and appropriate adjustments on the hyperparameters of the model are required.

*Figure 1.5* shows the subsets in which the experimental data available is split in case a cross validation method is applied and indicates the function each of them plays in models’ creation.

A picture containing text, screenshot, font, line

Description automatically generated

Figure 1.5 - Diagram of dataset splits with respective functions [22]

Crucially different from the training error, the error calculated on the validation dataset does not influence the alterations made on the model from the learning algorithm. This allows the neural network to use such validation dataset for testing the generalization capacity of the model also in the following epochs.

There are several types of cross validations, the simplest one is known as *holdout cross-validation*. This involves randomly subdividing the experimental data in two fixed training dataset and validation dataset for all the duration of the training phase [9].

### Regularization Methods

To solve situations of overfitting there are a series of solutions known as *regularization methods*. Apart from manual alteration of the hyperparameters and the architecture based on visual comparison among different generated models, one of the most widely used and easy to implement techniques to counter overfitting on the data is *early stopping.* A neural network which uses early stopping terminates the phase of training when the loss on the validation set stops improving after a programmer-given number of epochs [23].

Another common way to avoid overfitting situations is using techniques whichinvolves the introduction of a penalty term in the loss function in order to discourage the model to fit the noise in the data. This penalty term has an effect on the final solution proportional to the penalty value inserted at its declaration. Common regularization types used are *L1*, also known as Lasso regression, and *L2*, known as Ridge regression [9].

# Principles of Aerodynamical Properties of Objects

## Lift and Drag Coefficients

In the natural word, when solid objects interact with viscous fluids several forces are exchanged. When the fluid involved is the air or a gas, the forces perceived by the object as result of such interaction are known as *aerodynamic forces*.

The aerodynamic forces can be explained in simplified manner with two different but equally valid theorems [24]:

* As *Newton’s third law of motion* states [25]:

*“If two bodies exert forces on each other, these forces have the same magnitude but opposite directions.”*

This implies that given a three-dimensional object moving in a three-dimensional environment occupied by a viscous fluid, due to the sum of forces acting on it, the moving object will experience a force.

In Figure 2.1 is shown how the upward force (lift) perceived by an airfoil can be viewed as reaction to the *downwash* (the deflection of the airflow downward) resulting from the interaction between the airflow and the airfoil. The lift and the force necessary to deflect the airflow have the same magnitude but opposing directions [26]. This is an example in which the Newton third law of motion is applied in order to explain an aerodynamic force.

A picture containing screenshot, graphics, line, design

Description automatically generated

Figure 2.1 - Schematic representation of the lift perceived by an airfoil as reaction to the force causing the downwash [27]

* *Bernoulli’s equation* is a physics formula which relates the pressure of a fluid with its speed and density. In *incompressible flows* (fluids whose density remains constant in time), considering the fluid’s potential energy variation as negligible, such equation can be represented as follows [28]:

Immagine che contiene testo

Descrizione generata automaticamente

Equation 1.1 - Bernoulli's equation

*p*i = pressure of the fluid at instant *i*

*V*i = speed of the fluid at instant *i*

*ρ =* density of the fluid

In accordance with such equation the *Bernoulli’s principle* states that an increase in the speed of a fluid occurs simultaneously with a decrease in static pressure or a decrease in the fluid’s potential energy [29].

According to such a theorem, since the fluid moves at different relative speeds along the various sides of the object, the latter experiences a pressure difference on various areas of its surface. This generates, according to an addition of Bernoulli’s principle, a resulting force in the direction of the pressure gradient [29]. Since the faster the object moves relative to the fluid the higher the pressure gradient is, this force increases proportionally to the speed at which the fluid appears to move relative to the object.

In *Figure 2.2* is shown how the pressure gradient between the upper and lower edge of an airfoil generates the lift force perceived by this.

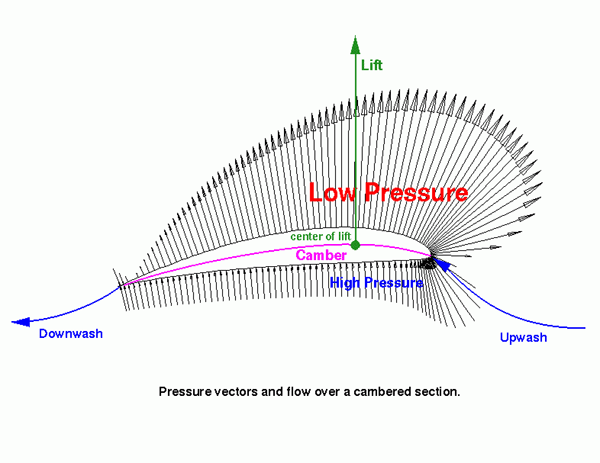


Figure 2.2 - Pressure vectors acting on the surface of an airfoil [30]

The resulting force acting on the object can be divided in three different vectors corresponding each to one of the physical dimensions. In aerodynamics, set a ground reference, the force vector parallel and in opposite direction to the object motion is known as *drag,* while the vector perpendicular to the object motion and pointing outward the ground plane is known as *lift*. The opposing forces to drag and lift are named respectively as *thrust* and *weight* [28]*.* In case the vertical component of the total aerodynamic force experienced by the object was negative, the force that is added to the weight in the downward direction is known as *downforce* [31].

The relative importance and relation of lift, drag, thrust, and weight/downforce forces acting on the objects depends on the application of use. For instance:

* in the aircraft and in wind energy industry maximizing lift and minimizing drag values from the wings and blades is crucial to achieve good fuel economy (in the case of the aviation industry) [32] and efficient wind’s energy use while avoiding high stress on the turbines (in the case of the wind energy industry).
* in the sports car industry high downforce and low drag on spoilers is preferable in order to improve traction and stability of the vehicles at high speeds while, at the same time, have minimum air resistance [31].

The various components of the resulting aerodynamic force perceived by the objects are always reported in scientific papers as coefficients. Dimensionless quantities like *lift and* *drag coefficients* (oftentimes written in their abbreviated forms Cl and Cd) are used to easily compare objects with different sizes and operating conditions [28].

## Airfoils and NACA Airfoils

*Airfoils* are two-dimensional geometrical shapes which represent flat sections of wings of airplanes, spoilers of sports vehicles or of the blades of wind turbines. Airfoils can have several shapes and characteristics and are used to study the overall behavior of objects constructed from such airfoils when these interact with a fluid.

Airfoils’ shapes are generally defined describing their characteristic attributes. These have specific nomenclature. The section of the generally narrow surface meant to face the flow of the fluid is known as *leading edge*. On the opposite side of the geometry, corresponding to the area immediately around point in which the long upper and the lower edge of airfoil meet, the *trailing edge* is located. The median line that crosses the airfoil from the leading to the trailing edge is called *mean camber line* (also camber for short), while the straight line that connects these two edges of the airfoil is reported in scientific papers as *chord line* (generally referred only as chord)*.* The maximum distance between camber and chord is known as *maximum camber.* The *maximum thickness* is instead the biggest spatial interval between the upper and the lower edge of the airfoil [28]. The various elements constituting the airfoils are shown in *Figure 2.3*.

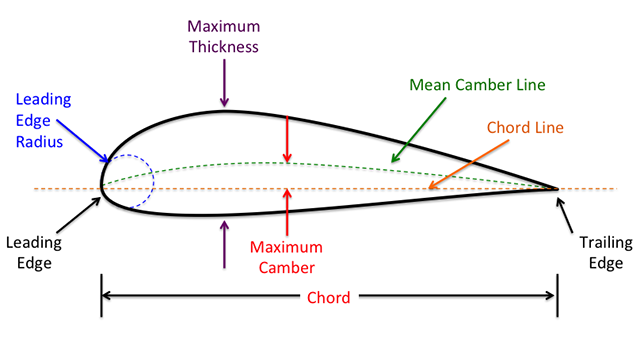


Figure 2.3 - Nomenclature of the various parts of an airfoil [28]

The first studies in this field were carried on by the National Advisory Committee for Aeronautics (NACA), a now defunct federal agency of the United States of America, which already in the late 1920s started to develop the first airfoils’ shapes who would be further refined by the Langley research Center and the United States Air Force [33].

A well-known series of airfoils engineered by the federal agency is known as 4-digits NACA airfoils. The shape of these airfoils is entirely described using the 4-digits code that identifies them. The parameters that define the geometry of the airfoils are the maximum camber, the location of this in the airfoil and the maximum thickness [34]. All these attributes are put in relation with the airfoil’schord, in particular:

* the first digit of the code specifies the maximum camber of the profile in percentage of the airfoil’s chord,
* the second digit defines the location of the maximum camber point in percentage relation with the airfoil’s chord and divided by 10,
* the third and the fourth digits indicate the maximum thickness of the airfoil, in percentage relation with the airfoil’s chord.

Based on these parameters the different 4-digits NACA airfoils are categorizable in regard of their characteristics. In this context, there is a specific terminology based on the camber of these geometries [34]:

* *Symmetrical airfoils*: these are airfoils in which the upper and lower edge of the geometry are mirrored along the chord. As a result, the chord and the camber line of the airfoil are superimposed. Symmetrical airfoils are characterized by having “00” as first two digits of the code.
* *Cambered airfoils*: airfoils in which the upper and the lower edge are asymmetrical. In extreme casesthe chord can run partially outside the airfoil.

In the airfoils the aerodynamic force, resultant of the several lift and drag forces acting on the geometry, can be viewed as acting on a single point of the object, known as *center of pressure* of the airfoil. In such point the torque given by the aerodynamic forces, called *pitching moment*, is equal to zero [29]. In symmetric airfoils the center of pressure is fixed at a quarter of the chord from the leading edge while in cambered airfoils this varies according to the inclination assumed by the airfoil (a.k.a. angle of attack, explained in detail in the following subchapter) [28]. A visualization of the force vectors and the center of pressure in a general cambered airfoil are shown in *Figure 2.4.*

Immagine che contiene diagramma

Descrizione generata automaticamente

Figure 2.4 - Airfoil schematic showing force vectors and center of pressure [35]

## Angle of Attack and Stall Angles

The angle that lies between the chord of an airfoil and the vector parallel to the wind apparent direction is known scientifically as *angle of attack* (AOA or alpha).

At a certain AOA, defined by the air’s *Reynolds number* (dimensionless quantity which describes the flow conditions, influenced by the relative velocity of the air and the airfoil, the size of the airfoil, and the kinematic viscosity of the air) as well as the airfoil’s design and surface roughness, airflow separation from the airfoil occurs. This particular condition is known as *stall* and the adoption of AOAs above the stall point causes significant effects on the aerodynamic forces experienced by the airfoil. This is clearly noticeable comparing the lift coefficients of an airfoil before and after stall has occurred: while at AOAs in proximity of the stall tends to slowly increase, beyond the stall angle this suddenly plummets [28]. The drag force is also highly affected by stall, resulting in an exponential increase in the related coefficient at higher airfoil’s inclinations. The mentioned effects of the stall on an airfoil are observable in *Figure 2.6*.

Generally, the stall occurs at lower AOAs the higher is the Reynold number of the fluid. Indeed, at greater relative speeds, the air experiences a more turbulent flow causing more *near-surface shear* (also referred to as *skin friction*), a phenomenon which makes airflow over the airfoil’s surface unsteady. High near-surface shear increases the skin friction experienced by the airfoil as well as the probability to induce flow separation at high angles of attack, causing a stall situation.

Flow separation is the main factor that leads to the increased drag and lowered lift experienced by the airfoils at angles of attack beyond the stall angle. Indeed, flow separation causes a large low-pressure turbulent wake in the section of the airfoil in which the airflow is separated, causing the insurgence of a new force called *pressure drag* which acts on the airfoil in the direction parallel and opposite to the airflow and which adds itself to the other drag forces experienced by the airfoil [28]. An example of turbulent wake of an airfoil in stall is observable in *Figure 2.5*.

Immagine che contiene grafico

Descrizione generata automaticamente

Figure 2.5 - NACA0012 airfoil in sub-stall (left image) and stall conditions (right image) [36]

Due to the known inefficiency and potential danger of this state, in modern civil airplanes complex adjustable wings help the vehicle to fly always at sub-stall conditions and, during aviation training, pilots are instructed to prevent stall conditions and recover control of the aircrafts in case the stall angle was overcome [37].

## Lift-to-Drag Ratio

Since, as previously said, the main goal of the airfoils in several industries is to maximize lift incurring in the minimal drag. For this reason, the *lift-to-drag ratio* (L/D ratio), which is the relation between lift and drag forces acting on the airfoils (or more generally an aerodynamic body) at a specified AOA and represents the aerodynamic efficiency of an airfoil given an inclination and the border condition, has a crucial importance for workers in the field [32].

In particular, *L/D ratio/AOA graphs* are analyzed by constructors of aircrafts and other vehicles as well as producers of turbines to have a complete picture of the behavior of the wings/blades in different spatial configurations and in different wind conditions. In the case of civil airplanes, for instance, an airfoil with good L/D ratios at AOAs commonly used by this type of aircrafts allows the planes to fly their routes minimizing energy use and, thus, have better fuel economy.

The characteristic arch-shaped curve present in L/D ratio/AOA graphs, result of the different evolution of the lift and drag coefficients of the objects at different AOAs, is clearly observable in *Figure 2.6*.

Immagine che contiene grafico

Descrizione generata automaticamente

Figure 2.6 - Lift coefficient, drag coefficient and L/D ratio of a generic airfoil [38]

# Basics of Computational Fluid Dynamics

## Definition of CFD

*Computational fluid dynamics* (CFD) is a branch of fluid mechanics that uses numerical analysis and data structures with the aid of computers to analyze and solve problems that involve fluid flow. In order to do so, virtual representations of the environment to study are created and used to perform the fluid dynamic calculations [39].

Navier-Stokes equations are highly complex partial-difference equations which define the behavior of many single-phase fluid flows. The CFDs generate the simulations on the models solving the Navier-Stokes equations of such systems through the application of specific numerical methods [40].

## CFD Preprocessing

The geometry and physical bound of the problem are created with the help of computer aided design (CAD).

When the model simulated is an airfoil conceptualized for the wing of an aircraft, it is crucial to create an environment which minimizes the *ground effects*. These are the aerodynamic forces perceived by the airfoil due to the pressure variations of the air that are the result of the interaction of the fluid flow with the shape of the environment’s external boundary [41]*. Figure 3.1* shows how the flying distance from the ground of a helicopter influences the induced airflow. In such an example the ground effects result in an increased lift perceived by the aircraft.

A picture containing text, drawing, design, illustration

Description automatically generated

Figure 3.1 - Airflow patterns induced by a flying helicopter at different distances from the ground [42]

After the geometry in CFD simulation it is fundamental to choose a proper *mesh*. This is the name with which it’s called the process of subdivision into cells of the space to simulate. The mesh of an CFD model can be either *structured* or *unstructured* according to the presence or absence of a recurring pattern in the partition of the whole spatial domain. For each cell created the turbulence model is applied and subsequently the mean aerodynamic values of the enclosed area are calculated. Generally, the higher is the cell density the more precise the results tend to be since there is a lower approximation of the simulated area, however, this will result in longer computation times [43].

The *boundary layers* are the thin regions of space which immediately surround each surface present in the fluid domain. In these regions the pressure of two points in space can change enormously also at relatively low distances, influencing significantly the results obtained in a simulation. Therefore, in order to achieve high quality predictions, it is necessary to have good approximations of the behavior of the fluid in the boundary layers of the system. The *Y+ value* is a dimensionless parameter, dependent on Reynolds number of the fluid, the fluid density and dynamic viscosity, and the object geometry which is used in CFD programs to define the size of the cell layers bordering the object surfaces [44].

Once set the geometry and the mesh of the model it’s necessary to adjust the physical modeling. This comprises several steps, like the insertion of the fluid viscosity, the object surface’s material, and fluid’s relative velocity to the object.

One of the crucial components to select is the *turbulence model*. Turbulence models is the class of mathematical models and numerical methods that are added to the Navier-Stokes equations to approximate the behavior of turbulent flows in the system. There are multiple types of turbulence models used in CFD programs, each of which has intrinsic characteristics and qualities that, depending on the context of use, can be both advantageous or disadvantageous to the efficiency and reliability of the simulation. These are nonetheless categorizable in two big branches, the more simple but less computational demanding *Reynolds-averaged Navier Stokes* (RANS)and the more complex but heavier to run *Large Eddy Simulation* (LES) turbulence models. The difference between these two groups of turbulence models is visible when highly turbulent steams are modeled [45]. How it’s observable in *Figure 3.3*, the higher degrees of freedom of the LES models allow a more faithful simulation of the natural behavior of the fluids compared to RANS turbulence models.

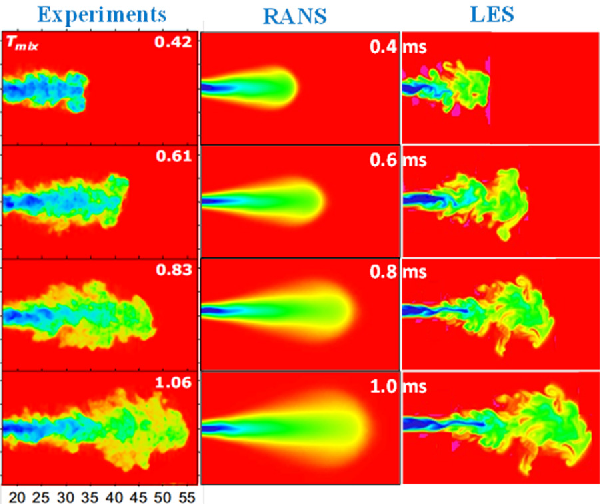


Figure 3.2 – Highly turbulent flow recreated using RANS and LES models [46]

The *Spalart-Allmaras* *model*, a turbulence model commonly used in CFD models, belongs to the RANS models’ branch. Despite its relative simplicity compared to other turbulence models, some of which are still categorizable as RANS, the Spalart-Allmaras model is still widely used in the field due its precise predictions on fluid flows with low-medium Reynolds numbers and its low computational demand. Its relative simplicity compared to the other models derives from the fact that it adds to the Navier-Stokes equations only an additional equation to model the turbulent viscosity of the flow. Indeed, the other turbulence models capable of modeling turbulent environments require two additional equations, increasing considerably the processing time as result [47].

*Solution methods* are a collection of parameters that are used to solve the Navier-Stokes equations in the simulation. These comprise settings on the pressure-velocity coupling, the spatial discretization, and the transient formulation [47]. The choice of the solution methods to implement depend on the specific problem to solve, the accuracy and precision desired and the available computational resources.

*Boundary conditions* define the properties and the behavior of the fluid in the proximity of the boundary layers. During this step, as an instance, the inlet and the outlet of the fluid, as well as contact walls in the system are defined [47].

## Simulation and CFD Postprocessing

When the preprocessing phase is concluded, the simulation is run. During this step, the Navier-Stokes equations are solved with the selected numerical methods, discretizing them in space and time and obtaining systems of algebraic equations that can be then computed iteratively. The obtained numeric solutions of each iteration are confronted with the solution obtained in the previous iteration and the difference between these values gives the *residuals,* values which assess the accuracy and precision and convergence of the numerical solution. The simulation proceeds until convergence is met by obtaining residuals values that satisfy a pre-defined convergence criteria on the simulated data or, in case this event doesn’t occur, when a determined maximum number of iterations is reached [47].

Once the simulation is completed, it is possible to analyze, visualize, and extract the results obtained on the CFD model.

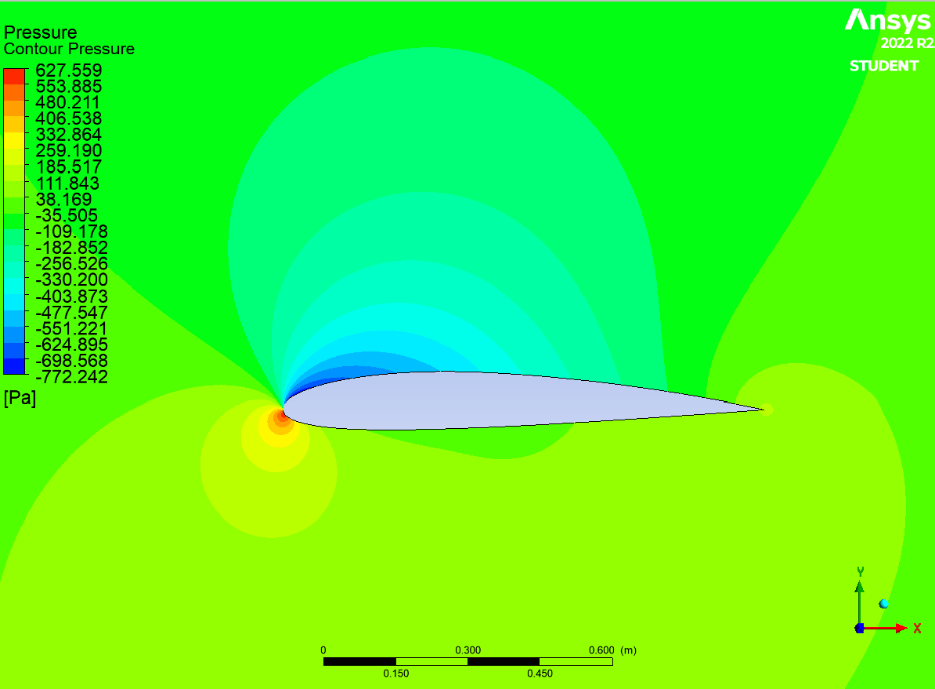


Figure 3.3 – CFD model simulating the pressure of the air around a NACA2412 airfoil at AOA=4° and Re=2.2e6 (image generated using the program Ansys Fluent)

# Implementation Process

## Preparatory work

### CFD Simulations

How previously mentioned, there is limited experimental data in real wind tunnels easily available for 4-digits NACA airfoils, both in number of airfoils analyzed as well as number of angles of attack and Reynolds number taken into consideration. For this reason, it was necessary to acquire a sufficiently large set of representative and reliable lift and drag data of airfoils which could were used as training and validation data for the neural network. This data was obtained through the simulation results obtained by CFD models of the various airfoil geometries on ANSYS Fluent, a reliable and professional software product owned by the American engineering simulation software company ANSYS. In order to validate the data generated through the simulations a comparison between them and the official NACA documents was carried forth whenever possible.

With the assistance of an airfoil plotter, 200 two-dimensional coordinate points constituting each airfoil geometry were generated by solving the mathematical function describing them. These were later divided and reported on different Excel files according to the airfoil they described. The length of the chord of the airfoils generated is 1 meter.

The geometry of the surrounding environment of the airfoils was designed taking as examples several geometries adopted by NASA’s Langley Research Center at fixed AOAs. In order to limit the ground effects resulting from the relative position of the airfoil in the simulated environment, it was decided to keep the airfoil in a fixed location with the chord parallel to the x-axis and, in each simulation, automatically modify the direction of the airflow to replicate the behavior of the airfoil at the given AOA. In order to limit the pressure variations related to the airflow direction, the design found on the NASA document was remodeled to avoid the exchange of any perpendicular force between the air and the external boundary of the model (except the outlet) at any AOA tested. The exact measures of the final design were selected through a careful process of trial and error by comparison of the simulated data with the experimental wind tunnel’s ones.

The model’s geometry and the airflows’ speed and direction in the fluid domain are visible in the screen captures shown in *Figure 4.1*.

Immagine che contiene diagramma

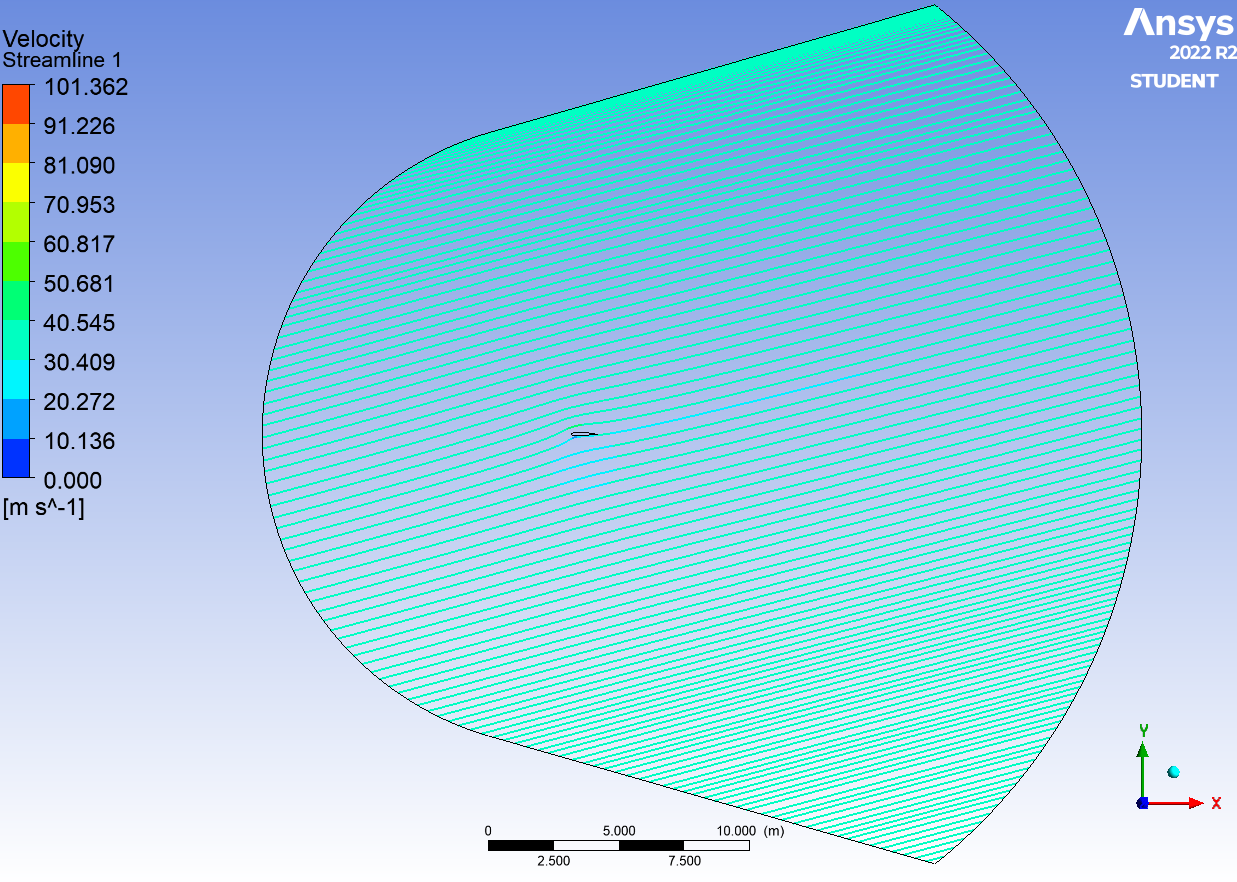
Descrizione generata automaticamente

Figure 4.1 - Airflows in the fluid domain at AOA=-2° and AOA=15° (image generated using the program Ansys Fluent)

The geometry of the fluid domain can be mathematically described as a horizontal asymmetric stadium. This particular shape was obtained by connecting with two tangents of 16°, respect to the x-axis, two circles with their centers lying inside the airfoil shape and with respective diameters of 12 and 21 meters.

The mesh of the CFD mode picked was designed with the following principles in mind:

* create a multi-layered near-wall mesh refinement around the airfoils with a Y+ value ≃1 in order to have precise predictions of the boundary layer behavior around the airfoil.
* avoid big size discontinuities among the several cells composing it with the purpose of not mixing partial calculations with a different level of approximation into the final results.
* have a structured mesh in order to accelerate the process of mesh generation and resolution.

The final designed mesh counts 238920 quadrilateral cells. Of these, 1320 directly border the airfoil’s surface and have a thickness of ≃11 μm, that returns a Y+ value in the desired range given the system state. Beginning from the airfoil surface the cells expand outward with a growing rate of 1.2 until they reach the extremities of the CAD model. Discontinuities were corrected through use of biases and adjustments on the number of divisions on the mesh sections in a process of edge sizing. A closeup of the created mesh is visible in *Figure 4.2*.

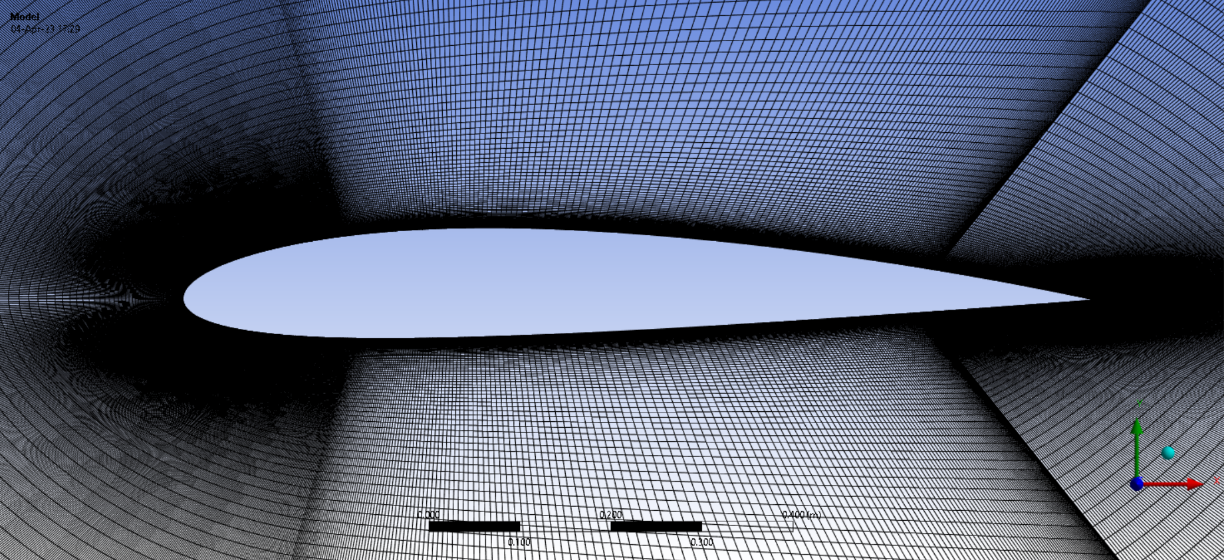


Figure 4.2 - Mesh in proximity of the NACA2412 airfoil (image generated using the program Ansys Fluent)

As relative velocity between the fluid and the airfoil a value of 32 m/s (0.094 Mach) was selected. Indeed, given a dynamic viscosity of 1.7894e-5 kg/(m\*s) and a density 1.225 kg/m^3 of the air as well as a unitary characteristic linear dimension, with the aforementioned velocity the fluid has a Reynolds number of approximately 2.2e6, for which NACA experimental values to compare the simulations results are available.

After testing different turbulent models and confronting the obtained results with wind tunnels data, the Spalart-Allmaras model was selected to perform the CFD simulations. Such a model was chosen since it returned lift and drag coefficients with accuracy similar to the ones generated using other turbulence models available on the program, but with considerably reduced processing time.

For the simulation phase, the default solution methods as well as other hyperparameters of ANSYS Fluent were kept unchanged.

The simulation process was configured to automatically conclude whether the continuity residual would have reached a value below 1e-6 or, in case this wasn’t achieved, whenever the number of iterations would have reached 300.

This whole process was automized to be repeated on 28 different 4-digits NACA airfoils for 21 consecutive integer AOAs (from -4° until 16° of the x-axis). In detail, the codes of the airfoils selected for the simulations are:

NACA0006, NACA0008, NACA0009, NACA0010, NACA0012, NACA0015, NACA0018, NACA0021, NACA0024, NACA1408, NACA1410, NACA1412, NACA2408, NACA2410, NACA2411, NACA2412, NACA2414, NACA2415, NACA2418, NACA2421, NACA2424, NACA4412, NACA4415, NACA4418, NACA4421, NACA4424, NACA6409, NACA6412.

These NACA airfoils were specifically selected for this project with the scope of teaching the neural network to recognize behavioral patterns of different categories of geometries in regard of their camber (symmetrical, cambered) and thickness (thick, thin) in sub-stall and near stall operation.

How shown in *Figure 4.3*, the obtained data were later graphically confronted with the experimental data available in order to validate the data generated. Results of airfoils with high AOA and lift and drag coefficients noticeably outside the experimental range due to post-stall effects were discarded in order to avoid that the error on such data influences the general quality of the predictions made by the deep neural network. The data of AOAs that theoretically should be slightly beyond the stall angle but didn’t differ significantly from the real values were kept in the training set in order to have a bigger data pool and test the effectiveness of the stall recognition mechanism later described.

Immagine che contiene grafico

Descrizione generata automaticamenteImmagine che contiene grafico

Descrizione generata automaticamente

Figure 4.3 - Experimental vs. simulated lift and drag coefficients of the NACA2412 airfoil

After the validation phase was completed, the number of lift and drag couplets acquired through the CFD simulation and not discarded was 572.

### Extraction and Transformation Operations of the Simulated Data

The data obtained through the CFD simulation reviewed and regarded as valid were transcribed on a Microsoft Excel file named “airfoils\_aerodynamic\_coefficients.xlsx”, reporting each of the result as a row structured in the following manner: airfoil’s code, AOA, drag coefficient, lift coefficient.

A python file, named “datasets\_preparation.py” was created in order to automatically convert the data stored inside the Excel file in a python list of tuples which contains, in ordered manner, all the information that the neural network uses either as training, validation or testing set.

In order to accomplish this, this python file imports the openpyx library which allows the direct management of data in Excel Spreadsheet.

Differently from the other files of the project, “datasets\_preparation.py” doesn’t host a main function but consists in a couple of functions that are called directly by the top-level code. This was made in order to refresh the training/validation and testing sets each time that the other files are executed importing it as a module. Indeed, the negligible execution time of the file doesn’t justify a two-steps refresh process of the data which would involve running “datasets\_preparation.py” as main program each time the data in the excel file are modified before running the other files of the project.

The first function called by the program is “excel\_to\_training\_set”. This takes the Excel workbook as input, transfers the data from the Excel sheet “Training Set” to a local variable and in an iterative process arranges the information of the aforementioned variable into a list named “training\_set”.

This iterative process was built with the purpose of collecting in an ordered way all the data connected to an airfoil code in a unique tuple inside the list. This approach was chosen in order to easily localize mistakes in the coding process, to facilitate the recognition of the stall point in the testing phase, and to effortlessly extract the airfoils codes without repetitions in the later explained “max\_lift\_drag\_ratio.py” file. As a result, each of the tuples in “training\_set” contains two elements: the first is the airfoil code, while the second is a nested list of tuples ordered by increasing AOA. Each of these inner tuples contains an AOA and the lift and drag coefficients obtained by the airfoil geometry with which they are linked to in the outer tuple when this is positioned with such AOA. The structure of “training\_set” is shown in *Figure 4.3*.

Immagine che contiene testo

Descrizione generata automaticamente

Figure 4.4 - training\_set structure

A function similar to “excel\_to\_training\_set”, named “excel\_to\_testing\_set”, is used to extract the data from the Excel sheet “Testing Set” and transfer them to a local variable named “testing\_set” using the same iteration process explained above.

At the termination of the iterative process the list of tuples stored in the local list “training\_set” is returned to the caller of the function, which in this file is a global variable also named “training\_set”, while the local list “testing\_set” is assigned to the global variable “testing\_set”. The partition of the training and validation sets is performed subsequently during the definition of the training phase hyperparameters.

Of all the 572 airfoil-AOA-drag-lift quadruplets obtained through CFD simulations, 528 (92.3%) were assigned to the Excel sheet “Training Set” to work as training and validation data for the neural network model, while 44 (7.7%) were assigned to the “Testing Set” sheet to be included in the testing set. This proportion replicates similar ratios to the ones used in other scientific papers.

The quadruplets chosen for the testing set belong to 4 different NACA codes, respectively: NACA2412, NACA4415, NACA2414, NACA0012. These geometries were selected in order to analyze the model behavior predicting coefficients on airfoils with different camber and thickness.

While the first two airfoil geometries in the above list have only one row each written on the “Testing Set” excel sheet, the last two have all their 21 experimentally obtained quadruplets fed to the testing set. This was made to both understand the preciseness of the model in making predictions on the entire spectrum of AOAs of airfoil geometries do not present in its training set, as well as to test the stall recognition mechanism implemented.

## Programming of the Deep Neural Network

Terminated the preparatory phase of the project, the programming of the actual deep neural network has taken place. The python file containing the implementation of the neural network model design and the hyperparameters necessary for its generation was named “deep\_neural\_network.py”.

In accordance with good programming practices, instead, the part of the program with the task of testing the neural network models generated was written on a different file, named “test\_models.py".

### Imports

In order to facilitate the process of programming the neural network, the open source neural network framework Tensorflow (version 2.11.0) was imported. Keras, a sub-library and Python interface of Tensorflow, allowed the use of the wide range of specific neural network modeling functions of the neural network framework in the Python programming language.

Keras is actively utilized in the definition of the neural network structure, as well as for the optimizer selection and implementation of overfitting prevention techniques like regularizers and early stopping.

In order to organize the input and output datasets of the neural network in array structures the NumPy library was imported.

Finally, the list of tuples contained in the global variable “training\_set” in the file “datasets\_preparation.py” is also imported. This was made to allow the program to have the airfoils data necessary for the neural network training and validation.

### Main Function call and Features Extraction

Following the import commands, the main function is executed. Indeed, the rest of the program inside the file “deep\_neural\_network.py” has been all enclosed in this function. This was made to ensure that in future developments the time-consuming process of generation of a new deep neural network model is performed only when the file is run as the main program.

The first lines of the implementation have the task to acquire from the list of tuples the features which the neural network model will use as input and output control during the training phase.

This is accomplished through a for cycle for each airfoil present in the list of tuples “training\_set” and whose procedure is divided in the two following steps:

* First, the digits of the current airfoil code of the cycle are subdivided into 3 different variables, each of which represents respectively the maximum camber, the maximum camber location, and the maximum thickness.
* Then, the series of AOAs, lift and drag coefficients incapsulated inside the list of tuples related to the airfoil code are extracted collectively and assigned to local lists.

### Training/Validation Dataset Creation

Remaining in the previously mentioned for cycle, a new nested for cycle is called. For each nested tuple in “training\_set” linked to the airfoil currently being analyzed in the outer for cycle, the features that were retrieved are appended to 4 lists.

These lists represent respectively the input and control output sets of the two MLPs that will have the task to make predictions of the drag and lift coefficient. The input sets of the two MLPs are similar, each containing the three airfoil-code-connected variables and the AOAs, however the input dataset meant for drag prediction additionally contains the lift coefficient values for reasons later clarified in this paper. The output sets contain respectively the coefficients of the physical force coefficient they are meant to predict.

All the input and output sets are temporarily assigned to the python lists containing the training sets of the lift and drag MLPs. The data meant for the validation set will be physically split from the training data during the training phase of the model.

### Architecture Definition

Concluded the phase of separation of the training and validation sets, the structure of the deep neural network model is declared. Before choosing the final design of the network a long experimentation phase analyzing the predicted lift and drag coefficients results obtained by testing different typologies of structures with various amounts of hidden layers and artificial neurons per layer, as well as with several classes and weight values of activation functions and regularizers.

The structure type that undeniably returned the best results during this procedure was a deep feedforward neural network design constituted by two interconnected MLP subunits, returning respectively the predicted lift and drag coefficients of the airfoil geometries tested at the desired AOAs and in which the forecasted lift outputs of the lift predicting MLP serve as additional input feature for the other MLP subunit.

Once the baseline architecture of the deep neural network was selected, it was performed an intensive process of testing of the deep FNN using different number of layers as well as various artificial neurons arrangements and configurations for each of the composing MLP subunits. Analyzing the predicted aerodynamic coefficients of the several candidate structures and confronting them with the expected values, it was found the configuration that maximized the accuracy and precision on the outputs. Such architecture had both the MLPs subunits consisting of 4 hidden layers but, while the artificial neurons in the hidden layers of lift MLP were arranged in a 32-16-16-16 configuration, the drag MLP had instead a 64-32-32-32 configuration. Such neural network structure, visible in *Figure 4.3*, was thus selected as architecture for the model to be generated.

Immagine che contiene diagramma

Descrizione generata automaticamente

Figure 4.5 - Graph of the deep neural network implemented

In order to utilize the neural network to its full potential, a leaky ReLU activation function with an alpha value of 0.3 was selected for all the hidden units constituting the deep FNN. Differently, for the node belonging to the output layers of the MLPs, in order to observe the full scale of the predictions, the identity function was preferred.

Analyzing the predictions and observing a slight difference between the loss values of the training and validation set it was decided to add a L2 regularizer with a weight of 0.01 on the first layer of each MLP of the system. This proved to reduce the difference between the two loss values, leading to overall better predictions in the testing phase.

### Training Phase Set Up

Each of the models was compiled using an ADAM optimization algorithm on a training loss calculated through the mean squared error loss function. ADAM was selected for this project after demonstrating faster convergency than the other optimization algorithms that were considered. After multiple learning rates were tested, it was decided to keep it to the default Keras value of 0.001 for both the MLPs.

The fit function initiates and runs the process of training of the model. In this section the input list and the output list were passed as arrays and the maximum number of epochs as well as an early stopping mechanism were declared. The iterations were configured to reach a maximum of 500, that during the numerous tests runs of the training phase were hardly reached due to interruptions imposed by the early stopping. For both the MLPs of the deep FNN, this mechanism was configured to halt the process of training of the model once the validation loss didn’t assume a new lowest value for 30 backpropagation iterations in a row.

During this phase the data is also divided between the training set and the validation set. Indeed, the parameter “validation\_split” in the fit function enacts a holdout validation method. With such addition, the function performs, during the first epoch, an automatic random subdivision of the series of tuples stored in the input and output lists, in training and validation datasets. The proportion of tuples assigned to each of the sets depends on the float value between 0 and 1 assigned to “validation\_split”. In this project such value was selected to be 0.1, meaning that the validation set is composed of 10% of all the data in the input and output lists, equivalent to 57 tuples each. This ratio between the size of the validation and training set was selected by imitation of precedents and through trial and error testing.

Terminated the process of training of the deep feedforward neural network model, the two MLP constituting it are saved separately on two hierarchical data format files in order to proceed to the process of testing the predictions made by the generated model.

### Testing Phase

In the file “test\_models.py”, the deep neural network models generated in the training phase are used to calculate lift and drag predictions for each airfoil-AOA data tuple present in the preconfigured testing set. Subsequently, using the experimental data present in such tuples, along with the neural network generated values, the percentage error on each of the coefficient is calculated. The results generated help to analyze the preciseness of each model and, through critical thinking, find ways to improve the quality of the predictions by properly adjusting the hyperparameters in the neural network definition file.

The first lines of the code serve to acquire all the necessary tools in order to perform the various tasks of the file:

* Libraries useful, if not required, for the programming of the file such as NumPy, Tensorflow, and time are imported.
* The file “stall\_recognition.py” is imported with the purpose to, through the use of its functions, allow the calculation of the percentage errors between the predicted and the experimental data as well as to recognize situations in which such predictions cannot be performed by the model because at risk of be inaccurate due to post-stall point effects.
* The experimental drag and lift coefficients of the testing set are acquired importing the global list of tuples “testing\_set” from the “datasets\_preparation.py” file.
* The lift and drag MLPs models constituting the deep neural network to test and stored on hierarchical data format files are loaded on “lift\_model” and “drag\_model”.

The main function contains a nested for loop. The outer one cycles through the airfoils present in the “testing\_set” list, while the inner one iterates the enclosed process going sequentially through the list of tuples linked to the airfoil currently analyzed in the outer loop. In the initial state of the main function and in any other case that the global Boolean function “suspected\_stall” is assessed as false in the inner loop, the function “predict\_lift\_drag” is called by the program.

How the name of the function suggests, “predict\_lift\_drag” has the task of instructing the tested neural network model to generate predictions of the drag and lift coefficient of the current airfoil geometry in the outer loop of the main function at the AOA extracted from the current tuple in the inner one. These coefficients are obtained by calling the “predict” function of the MLP models and passing it the necessary input data to perform the prediction.

The function “stall\_recognition” is then called from the homonymous python file. The purpose of this function is to recognize, through empirical testing, whether the considered AOA could lay past the stall point of such airfoil and thus assess whether the predictions made by the model could be regarded as reliable in such configuration. In this function two additional predictions are made respectively on the drag and lift value of the airfoil, but, this time, at an AOA of 2 degrees lower compared to the one passed as input parameter. These are then used in an if-then-else construct which checks whether the difference in value between the two drag coefficients predicted is equal or greater than 15% of the drag value of the airfoil configuration with higher AOA while, simultaneously, the two lift coefficients differ for less than 8% of the value of steepest one. According to the fulfillment of these conditions, the function terminates returning a certain Boolean value, which is then assigned to the previously mentioned variable named “suspected\_stall”.

It was decided to analyze predictions with an airfoil’s inclination difference of 2 degrees in order to minimize the influence of possible fluctuations between subsequent AOA predictions in proximity to the stall point. The values of respectively 15% and 8% difference on 2-degrees-distanced drag and lift coefficients were set after observing them as an approximated threshold at which stall occurs on the available scientific papers of the airfoils analyzed.

If the variable “suspected\_stall” is assessed to be true, the AOA analyzed could potentially be beyond the stall of such airfoil geometry, therefore the calculation of the percentage error on the experimental data doesn’t occur since it has a high probability to be inaccurate when confronted with real wind tunnel data. The program would thus print a message at screen informing the user about the event and it would, in the main function, halt the reading of the tuples of the current airfoil, so not to perform any predictions on such airfoil geometry at steeper AOAs.

Differently, if the conditions of the if-then-else construct in the “stall\_recognition” function are not realized, the AOA is not considered as likely to be beyond the stall point of the airfoil, thus the function “calculate\_percentage\_error” is called for both the lift and drag coefficients. This function has the task of returning the percentage errors given the predicted and the experimental data as input parameters. Finally, the results are printed on the terminal to allow visual analysis.

## Max L/D Ratio Search

As previously mentioned, knowing the lift and drag coefficients of the airfoils is useful in certain industries to study their L/D ratios and at which conditions this proportion reaches its maximum. For this reason, the file “max\_lift\_drag\_ratio.py” was implemented.

The program has the task of testing multiple airfoils and returning the wing profile and the AOA that achieves the maximum L/D ratio. The aim of the program is to test airfoils that can also not figure among the geometries that were modeled in CFD. In order to guarantee that the returned data were valid, the file was at first tested using solely the available airfoils and later, when its predictions were deemed reliable, another prediction was made adding 6 new 4-digits NACA airfoils to the list of airfoils to test.

In the case of the prediction with solely airfoils used in the CFD simulations, the result was confronted with the expected L/D ratios present in the fifth column of each row of the excel sheets in the file “airfoils\_aerodynamic\_coefficients.xlsx”.

The respective codes of the new geometries tested in the second prediction are: NACA2406, NACA3412, NACA4312, NACA4420, NACA6406, NACA7409. These airfoils have camber, maximum camber location, and thickness characteristics not considerably different from the ones of the airfoil geometries with which the deep neural network was trained and tested, meaning that their predicted L/D ratios would likely be reliable.

As already seen for the other files described in paper, with the aim of implementing the program efficiently, several import commands are declared at the beginning of the file:

* The library Tensorflow is necessary to enable the use of the MLP models generated and imported into the program from the respective hierarchical data format files.
* The library NumPy is used to transform the input parameters of the “predict” functions of the models in arrays and insert a series of equally distanced values in the list selected to contain the AOAs to test.
* The file “stall\_recognition.py” is imported with the purpose of using its functions to avoid the calculation of inaccurate L/D ratios for airfoil configurations with an AOA beyond the stall point of such airfoils.
* The training/validation and testing sets are imported from the file “datasets\_preparation.py” in order to acquire the non-user-added airfoil codes on which to measure the L/D ratios.

In the first lines of code of the main function, the empty list “naca\_codes” is declared. This is then instructed to collect all the user-given airfoils and the airfoils in the training/validation and testing sets not present in such user list.

The list of AOAs on which the predictions are performed contains 30 evenly distanced and numerically ordered computer-generated values comprised in the interval between -4° and 16°. For each airfoil geometry present in “naca\_codes”, in a sequence from the lowest to the highest AOA value of the aforementioned list, the neural network model calculates the lift and drag coefficients prediction and the imported function “stall\_recognition” is called. If the airfoil configuration is not suspected to be in stall, the predicted values are appended on two separate lists and the cycle continues to the following AOA in the list, otherwise, the program terminates immediately the generation of predictions for such airfoil.

Then, for each airfoil geometry, the program calculates the L/D ratios dividing the equally indexed lift and drag coefficients contained in the respective two lists previously generated and assign them to a list. The highest ratio found in such list is thus allocated to the variable “airfoil\_max\_lift\_to\_drag\_ratio”. In case the value of such variable is assessed to be higher than temporarily absolute maximum L/D ratio of the previously analyzed airfoils, the value, along with the airfoil code and the AOA at which that specific L/D ratio is achieved, is recorded. In order to avoid interference between different tested airfoils, at the beginning of each iteration of the for cycle the lists containing the predicted drag and lift values are reset.

Finally, terminated the calculations on all the airfoils, the best performing airfoil, along with the maximum L/D value achieved and the optimal AOA, is printed on screen.

The result obtained in the first prediction was confronted with the experimental L/D ratios calculated on the F column of the “Training Set” and “Testing Set” sheets of the excel file “airfoils\_aerodynamic\_coefficients.xlsx”

Since the second prediction returned results different from the first, it was decided to validate such result by confronting the ratio obtained by the deep neural network model and the L/D ratios returned by the CFD simulation of the airfoil geometry which the model indicated as best performing.

# Discussion of the Obtained Results and Conclusions

## Results

The compile and execution time of a file depends on the computing power of the computer on which such processes are undertaken. In the case of this project, the process of compiling and executing the code of each of the files was performed on a Lenovo IdeaPad 3 laptop with an AMD Ryzen 5 3500U processor.

The mean time values shown in the following subchapters were calculated averaging 10 different compile and execution time measurements of each file.

### datasets\_preparation.py

Average compile and execution time of the file:7 seconds

### deep\_neural\_network.py

Average compile and execution time of the file: 2 minutes and 42 seconds

### test\_models.py

Average compile and execution time of the file: 1 minutes and 03 seconds

**Data obtained:**

The absolute and percentage error on drag and lift coefficients predictions performed by the model on the testing set are briefly explained in the following points of this subchapter and visually represented using histograms.

Additionally, the predicted lift and drag coefficients of the two airfoil geometries in which the whole range of AOAs were tested (NACA2414 and NACA0012) are represented using line graphs.

The numerical value of the predicted lift and drag coefficients, as well as the exact percentage and absolute errors on the expected results obtained are visualizable on the file “Test\_model\_results.xlsx”.

The program detected a potential stall situation for both the NACA2414 and the NACA0012 airfoils. These were signaled by the stall recognition mechanism respectively at 14° for the NACA2414 airfoil and 15° of angle for the NACA0012. After these detections, the program, as instructed in the code, halted the testing phase and didn’t return any prediction of the aforementioned airfoils with tilts greater or equal to such degrees.

* **Drag Coefficients**

The drag coefficients obtained by the model and their percentage and absolute error compared to the expected values are represented in the histograms in *Figure 5.1*, *Figure 5.2*, and *Figure 5.3*. From these charts it is observable the high accuracy with which the deep neural network was able to make predictions on all the airfoils present in the testing set in the range of AOAs from -3° to 10°. At these angles, indeed, the measured percentage errors on the expected data were always lower than 10%. Very remarkable are the predictions made by the model on the NACA2414 airfoil in the interval between -1° and 6°. These have indeed outstanding accuracy and precision, with all percentage errors approximatively equal or lower than 2% and reaching error values as low as 0.48% on 4°.

At steeper AOA than 7° both the absolute and the percentage error on the results increase considerably, in a proportional manner the higher the angle is. The highest percentage error value was 17.9%, obtained on the NACA2414 airfoil at an AOA of 13°.

Looking at the numerical values of the coefficients predicted, it clearly appears that the generated model tends to overestimate the drag force at steep AOAs. It’s also observable that the deep neural network wasn’t able to capture the slight increase of drag at decreasing AOAs during its training phase.

Figure 5.1 - Drag coefficients predicted from the deep neural network and expected values

Figure 5.2 - Percentage error measured confronting the drag coefficients predicted from the deep neural network and experimental data

Figure 5.3 - Absolute error measured confronting the drag coefficients predicted from the deep neural network and experimental data

* **Lift Coefficients**

The lift coefficients obtained by the model and their percentage and absolute error compared to the expected values are represented in the histograms in *Figure 5.4*, *Figure 5.5*, and *Figure 5.6*. Analyzing these charts, the deep neural network seems to return high quality predictions on AOAs from 1° until the stall point of the NACA2414 and NACA0012 airfoils is detected by the stall recognition mechanism. In this range the percentage error average 1% considering all the tested airfoils and reaches a minimum in NACA2414 at 7° of angle, in which a very negligible 0.008% error was calculated on the expected lift coefficient.

On the exact same AOA in which the lowest percentage value was recorded, the NACA4415 airfoil stands out for having a high absolute error in relation to the other airfoils even though this translates to an error of less than 4% in the percentage error histogram.

Also on the downside, the lift coefficients predicted on negative AOAs have swinging percentage errors, with increasing absolute errors the lower the AOA is. On the line graph representing the numerical values of the coefficient, however, this divergence between predicted and expected data is only slightly noticeable.

Worth of a particular mention is the apparently extremely high percentage error on the NACA0012 airfoil at 0° of angle. In this instance the ratio between the value of the lift coefficient predicted by the model and the expected data was found to be over 6266%. As a consequence of this, the top part of its percentage error histogram had to be trimmed for the sake of better data visualization of the other elements in the chart. Despite this gargantuan percentage error, the difference between the data is barely noticeable in the graph with the numerical values of the coefficient, while in the absolute error histogram the value measured is smaller than the one on other AOAs.

Figure 5.4 - Lift coefficients predicted from the deep neural network and expected values

Figure 5.5 - Percentage error measured confronting the lift coefficients predicted from the deep neural network and experimental data

Figure 5.6 - Absolute error measured confronting the Lift coefficients predicted from the deep neural network and experimental data

### max\_lift\_drag\_ratio.py

* **Results of the file using solely the airfoil geometries in the training/validation and testing sets**

Average compile and execution time of the file: 5 minutes and 26 seconds.

Obtained result: Airfoil = NACA6409, AOA = 4.276°, L/D ratio = 77.88051

Expected result: Airfoil = NACA6409, AOA = 5°, L/D ratio = 74.41603

Figure 5.7 - Experimental L/D ratios and predicted L/D ratio of airfoil NACA6409 at the same intervals

Percentage error between obtained and expected L/D ratio: 4.655%

Absolute error between obtained and expected AOA: 0.724°

* **Results of the file including the 6 additional airfoil geometries**

Average compile and execution time of the file: 6 minutes and 32 seconds.

Obtained result: Airfoil = NACA7409, AOA = 3.586°, L/D ratio = 82.35972

Result from CFD simulation: AOA= 4°, L/D ratio = 75.28264

Figure 5.8 - Experimental L/D ratios and predicted L/D ratio of airfoil NACA7409 at the same intervals

Percentage error between predicted and simulated L/D ratio: 9.398%

Absolute error between predicted and simulated AOA: 0.414°

## Analysis of the Data Achieved and Comments

Analyzing the results, the generated deep neural model seems to fulfill all the desired characteristics listed in the introduction section of this paper.

Timewise, it can be observed how the model is able to perform predictions on 28 airfoils at 30 different inclinations each in less than 10 minutes using a medium range laptop. As the L/D ratio prediction made with the additional airfoils suggests, the time needed to compile and execute the file increments proportionally with the number of airfoils to test. This time is on average 12 seconds for each airfoil, which is enormously different from the, always on average, circa 1 hour and half that would be needed to run the same number of CFD simulations for such scope. This instrument can therefore shorten considerably the amount of time necessary to acquire the lift and drag coefficients values of the airfoils and thus accelerate the research of more innovative types of airfoils.

The model is also lightweight. Indeed, summing the two hierarchical files containing the model and the python files required to run it, the total memory space occupied by the program is barely 173 kB. This value is more than 22 times lower than the almost 4 MB occupied by the XFOIL program. It should be also considered that XFOIL, as already mentioned in the introduction section of this paper, doesn’t directly calculate L/D ratios [48] and thus it is also not able to compare such value between multiple airfoils like the implemented program.

About the values obtained by the predictions, the accuracy and the preciseness of the results are always within a reasonable margin of error when confronted with the expected data. The ability of the neural network to identify the best performing airfoil, as well as the low difference in terms L/D ratio and AOA at which this maximum value is achieved in the prediction using the original 28 airfoils geometries, demonstrates the high reliability of the implemented project.

Addressing the predicted coefficients inconsistencies, most of the apparent criticalities observed are easily justifiable as negligible and result of the several approximations adopted in the various steps that led to the results. For the other issues found in the testing phase, even though none of these invalidate the overall reliability of the current model, some corrective and non-complex implementable solutions can be adopted to improve both the accuracy and the preciseness of future generated models. All the major criticalities, or apparent criticalities, are presented together with their eventual solutions in the following points:

Drag Coefficients:

* The high error on the drag predictions at high AOA is due most probably to the influence of simulated post stall drag values distant from experimental data that, due to the lack of wind tunnel data for some of the NACA airfoils, weren’t discarded and, as consequence, were erroneously incorporated in the training dataset of the model. An increase in the accuracy of such predictions could be achieved by modifying the stall recognition mechanism in order that it could be run on the file “datasets\_preparation.py” whenever a given airfoil has two rows in the “Training Set” sheet of the file “airfoils\_aerodynamic\_coefficients\_2.xlsx” in which the respective AOAs are at a distance of 2 degrees one from the other. In such a solution, the program would prevent the aggregation of data on the training/validation set whenever the mechanism recognizes a post stall situation.
* The increase in drag coefficients at negative AOAs is probably not modeled by the neural network because some airfoils with high camber present in the training set experience such increase in drag at AOAs lower -4°. Realistically, the model was influenced by the data of such airfoils during the training phase, leading to this inaccuracy in the predictions it makes. A solution to fix such a problem without increasing the complexity of the neural network could be inserting in the training data CFD simulated lift and drag coefficients at higher negative AOAs. The presence of these data would likely allow the neural network to make better predictions on airfoil configurations with AOAs under 0°.

Lift Coefficients:

* The extremely high percentage error in the NACA0012 airfoil at 0° of angle is the result of the very low expected data in such configuration. Indeed, theoretically, in symmetric airfoils at 0° of AOA the resulting lift force perceived by the object should be 0 [28]. Due to this characteristic of the uncambered airfoils, even a negligible amount of noise on the predictions of such configurations can result in enormous percentage error. This can be confirmed easily by looking at both the absolute error histogram and the coefficient values line graph.
* The high absolute error on the NACA 4415 at 7° of angle and on the other airfoil at low AOAs translate in relatively low percentage errors. As in the point presented above, a specific solution is therefore not required.

Looking at the L/D ratios of the several airfoils in the “airfoils\_aerodynamic\_coefficients.xlsx” file it is not surprising that the NACA7409 airfoil was found as best performing geometry. As it is observable in the graph present in *Figure 5.9*, in the conditions in which the airfoils are analyzed, there is a direct correlation between the maximum camber of an airfoil and the highest L/D ratio achievable.

Figure 5.9 - Relation between the maximum camber and experimental maximum L/D ratio on the airfoils at Re=2.2e6

This is explainable through Bernoulli’s principle. As it’s observable in *Figure 5.10*, the NACA7409 is classifiable as a highly cambered airfoil, in which the chord runs partially outside the airfoil. In such geometries, at low velocities and low AOAs, the airflow flows along the upper side at a faster speed than the other airfoil types. This generates a large low-pressure area above the airfoil and thus a high lift force that outweighs the characteristic high drag of this type of geometries, resulting in greater L/D ratios.

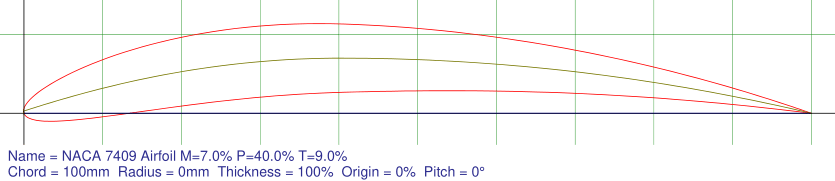


Figure 5.10 - Geometry of a NACA7409 airfoil (image generated on the website airfoiltools.com)

These aforementioned advantages of the airfoils with high camber rapidly disappear the higher the turbulence in the environment is. Indeed, given a subsonic relative speed between the fluid and the airfoil, the particular geometry of these airfoils causes the flow separation from the upper edge of the airfoil to occur at lower AOAs than the other airfoil types. As a result, the drag force perceived by highly cambered airfoils increase exponentially the more significant the turbulence in the fluid is [49].

Some post-WW2 single engines aircraft types like the Aeronca Chief and Aeronca Sedan used 4-digits NACA airfoils with high camber due to their efficiency and capacity to generate lift at low speeds [50]. However, most of these airfoils became soon obsolete for most uses since the aviation industry, already in the 40s, was able to develop airfoils, like the 6-series NACA, that allowed an efficient generation of lift at faster speeds and steeper AOAs compared to airfoil geometries like the NACA4412 adopted by the Aeronca’s planes [51]. Today aircrafts’ construction companies use mostly complex systems of adjustable wings in order to guarantee optimal L/D ratio during the whole flight [52]. Nonetheless, traditional 4 digits NACA airfoils with high camber are still seldomly used on low-speed planes like the close air support U.S. aircraft Fairchild A-10, whose wings are designed using both the NACA6716 and the NACA6713 airfoils [28].

# Product Backlog and Possible further Developments

Being this project for exclusive academic research and having limited resources at disposal, the project development was limited in scope to the goals mentioned in the introductive section.

However, with sufficient time and computational power, the project is easily amplifiable to more complex and specific needs, resulting in different possible ready-for-market products. This is possible thanks to the fact that the project was conceived using an incremental model.

The Scrum agile framework, which was adopted for such endeavor, subdivides the project into multiple tasks (known as *sprints* in Scrum) ordered in a *product backlog* using metrics like the relevance for the project and the estimated time needed for the implementation of the task.

Here are listed the sprints in the product backlog of this project which were not implemented in the final work:

* Implementation of the solutions presented in the section regarding the reflections on the results.
* Adaptation of the deep neural network to enable predictions also on 5-digits NACA airfoils.
* Creation of a file aimed at the extraction of the maximum weight-to-drag ratio for reversed airfoils.
* Inclusion of variations of 4-digits and 5-digits NACA airfoils with special nose features in the training, validation and testing sets.
* Deep neural network remodeling in order to allow the training on complex 1, 6, 7 and 8 series of NACA airfoils.
* Adoption of a multi equational turbulence model for the simulations of the airfoils aimed to find a more defined correlation between lift and drag coefficient values measured by CFD models and stall point of the airfoils.
* Gathering of training and testing data with different airfoil-air relative speeds with the purpose of allowing the identification of the airfoil that achieves best performance in different wind conditions.
* Comparison of the results of backpropagation with the ones of more recent learning algorithms like NEAT (NeuroEvolution of Augmenting Topologies) in order to uncover more effective and efficient methods to generate the deep neural network model.
* Overhaul modification of the script to adapt it to modern complex adjustable airfoil designs.
* Gather real-life lift and drag data of airfoils at high AOA to allow reliable results of airfoils’ behavior beyond the stall point.
* Implement a functionality of inverse design in which, given an input of desired behavioral characteristics, the program returns an airfoil shape which satisfies the parameters.

In this project, the sprints present in the product backlog are ordered considering multiple factors. The sorting positions are impacted significantly by the complexity of the tasks and the estimated time that these would have required for their completion considering the project budget and the computational power available. Another essential criterion that influenced the ordering of the sprints is the actual usefulness of the features acquired completing each of the tasks. This is made considering the state of the system following the accomplishment of the sprints that, taking into account also the other sorting factors, would be located in the backlog above the tasks.

# Acknowledgments

# Bibliography

|  |  |
| --- | --- |
| [1] | *MAE3253 - Introduction to XFOIL for Applied Aerodynamics,* Stillwater, Oklahoma: Oklahoma State University, 2019. |
| [2] | Cère-Aéro, "Flow5," [Online]. Available: https://flow5.tech/index.php/flow5/sample-page/xfoil-xflr5-flow5/. [Accessed 6 4 2023]. |
| [3] | "Direct numerical simulation," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Direct\_numerical\_simulation. [Accessed 11 April 2023]. |
| [4] | G. Novati, L. d. Laroussilhe and P. Koumoutsak, Automating turbulence modelling by multi-agent reinforcement learning, Nature Machine Intelligence, 2021. |
| [5] | "Scrum for one: A tutorial on adapting Agile Scrum methodology for individuals," [Online]. Available: https://www.lucidchart.com/blog/scrum-for-one. [Accessed 12 April 2023]. |
| [6] | J. R. Hansen, The History of the Langley Research Center, Washington D. C.: National Aeronautics and Space Administration, 1987. |
| [7] | E. R. Kandel and T. M. J. J. H. Schwartz, Principles of Neural Science, McGraw-Hill Professional, 2000. |
| [8] | R. A. Alzahrani and A. C. Parker, "Neuromorphic Circuits With Neural Modulation Enhancing the Information Content of Neural Signaling," in *ICONS 2020: International Conference on Neuromorphic Systems 2020*, 2020. |
| [9] | S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995. |
| [10] | B. Mwandau and M. Nyanchama, "Investigating Keystroke Dynamics as a Two-Factor Biometric Security," Datacamp, June 2018. [Online]. Available: https://www.researchgate.net/publication/325870973\_Investigating\_Keystroke\_Dynamics\_as\_a\_Two-Factor\_Biometric\_Security. [Accessed 26 May 2023]. |
| [11] | M. Maynard, Introduction to Artificial Neurons, Backpropagation and Multilayer Feedforward Neural Networks with Real-World Applications, 2020. |
| [12] | I. Basheer, "Selection of Methodology for Neural Network Modeling of Constitutive Hystereses Behavior of Soils," December 2002. [Online]. Available: https://www.researchgate.net/publication/229658224\_Selection\_of\_Methodology\_for\_Neural\_Network\_Modeling\_of\_Constitutive\_Hystereses\_Behavior\_of\_Soils. [Accessed 26 May 2023]. |
| [13] | D. Svozil, V. Kvasnicka and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems,* 1997. |
| [14] | L. Deng, D. Yu and J. Platt, "Scalable Stacking and Learning For Building Deep Architectures," in *ICASSP 2012*, 2012. |
| [15] | S. R, "ChatGPT Architecture Explained," 30 January 2023. [Online]. Available: https://medium.com/@sreedevr/chatgpt-architecture-explained-7fc0cbb7426c. [Accessed 11 April 2023]. |
| [16] | R. Parhi and R. D. Nowak, "The Role of Neural Network Activation Functions," *IEEE Signal Processing Letters,* vol. 27, 2020. |
| [17] | J. A. Snyman and D. N. Wilke, Practical Mathematical Optimization, Springer, 2018. |
| [18] | D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014. |
| [19] | G. Seif, "Understanding the 3 most common loss functions for Machine Learning Regression," 21 May 2019. [Online]. Available: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3. [Accessed 12 April 2023]. |
| [20] | B. Manela, "Deep Reinforcement Learning for Complex Manipulation Tasks with Sparse Feedback," January 2020. [Online]. Available: https://www.researchgate.net/publication/338569900\_Deep\_Reinforcement\_Learning\_for\_Complex\_Manipulation\_Tasks\_with\_Sparse\_Feedback. [Accessed 26 May 2023]. |
| [21] | G. C. Cawley and N. L. C. Talbot, "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation," *Journal of Machine Learning Research,* vol. 11, 2011. |
| [22] | S. Sorrentino, "Feature crossing per migliorare il nostro modello ML," Blexin, 14 January 2020. [Online]. Available: https://blexin.com/it/blog/feature-crossing-to-improve-our-ml-model/. [Accessed 26 May 2023]. |
| [23] | D. S. Goswami, "Introduction to Early Stopping: an effective tool to regularize neural nets," 9 August 2020. [Online]. Available: https://towardsdatascience.com/early-stopping-a-cool-strategy-to-regularize-neural-networks-bfdeca6d722e. [Accessed 12 April 2023]. |
| [24] | C. N. Eastlake, "An Aerodynamicist’s View of Lift, Bernoulli, and Newton," *The Physics Teacher,* no. 40, 2002. |
| [25] | S. T. Thornton and J. B. Marion, Classical Dynamics of Particles and Systems (5th ed.), Brooke Cole, 2004. |
| [26] | D. Halliday and R. Resnick, Fundamentals of Physics 3rd Ed., John Wiley & Sons. |
| [27] | M. Paetzold, *AirfoilDeflectionLift\_W3C,* Wikidata, 2018. |
| [28] | J. John D. Anderson, Fundamentals Of Aerodynamics 5th Edition, New York: McGrawHill, 2011. |
| [29] | L. J. Clancy, Aerodynamics, London: Pitman Publishing, 1975. |
| [30] | M. Chakraborty, "A Computational Study on two horizontally close sequential airfoils to determine conjoined pressure distribution and aerodynamic influences on each other," May 2020. [Online]. Available: https://www.researchgate.net/publication/281824169\_A\_Computational\_Study\_on\_two\_horizontally\_close\_sequential\_airfoils\_to\_determine\_conjoined\_pressure\_distribution\_and\_aerodynamic\_influences\_on\_each\_other. [Accessed 26 May 2023]. |
| [31] | S. McBeath, Competition Car Aerodynamics, Sparkford: Haynes, 2006. |
| [32] | N. Hall, "Beginners Guide to Aeronautics," Glenn Research Institute, NASA, 29 April 2022. [Online]. Available: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/. [Accessed 12 April 2023]. |
| [33] | B. Allen, NACA Airfoils, NASA, 2017. |
| [34] | I. H. Abbott and A. E. Von Doenhoff, Theory of wing sections: including a summary of airfoil data, Dover, 1959. |
| [35] | CFI, "CFI Brief: Angle of Attack as it relates to the Lift Coefficient," 11 August 2016. [Online]. Available: http://learntoflyblog.com/2016/08/11/cfi-brief-angle-of-attack-as-it-relates-to-the-lift-coefficient/. [Accessed 26 May 2023]. |
| [36] | K. Kuwahara, Director, [Film]. Japan: Institute of Space and Astronautical Science. |
| [37] | J. Donoghue, "Stop Stalling," 10 May 2011. [Online]. Available: https://flightsafety.org/asw-article/stop-stalling/. [Accessed 12 April 2023]. |
| [38] | C. Valeri, "Stalls, Deep Stalls, and What We Can Learn From The Air France Accident," Expert Aviator, 2 June 2011. [Online]. Available: https://expertaviator.com/2011/06/02/stalls-deep-stalls-and-what-we-can-learn-from-the-air-france-accident/. [Accessed 26 May 2023]. |
| [39] | R. Bhaskaran and L. Collins, "Introduction to CFD Basics," Cornell University, Ithaca, New York, USA, 2009. |
| [40] | "Numerical methods in fluid mechanics," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Numerical\_methods\_in\_fluid\_mechanics. [Accessed 11 April 2023]. |
| [41] | Z. Zaheer, K. E. R. Roy, G. S. Nair, V. Ragipathi and U. V. Niranjan5, "CFD analysis of the performance of different airfoils in ground effect," *Journal of Physics: Conference Series,* vol. 1355, 2018. |
| [42] | V. Hellin, "Helicopter Aerodynamics of Flight," Aeronautics Guide, [Online]. Available: https://www.aircraftsystemstech.com/2017/06/helicopter-aerodynamics-of-flight.html. [Accessed 26 May 2023]. |
| [43] | O. Zikanov, Essential Computational Fluid Dynamics, 2nd Edition, Wiley, 2019. |
| [44] | C. E. Brennen, Internet Book on Fluid Dynamics, Dankat Publishing, 2016. |
| [45] | IdealSimulations, "Turbulence Models In CFD," [Online]. Available: https://www.idealsimulations.com/resources/turbulence-models-in-cfd/. [Accessed 11 Aprile 2023]. |
| [46] | S. Som, P. Senecal and E. Pomraning, "Comparison of RANS and LES Turbulence Models against Constant Volume Diesel Experiments," in *ILASS Americas 24th Annual Conference on Liquid Atomization and Spray Systems*, San Antonio, TX, 2012. |
| [47] | ANSYS, "ANSYS Fluent 12.0 Theory Guide," ANSYS Inc., 2009. |
| [48] | "XFOIL, Subsonic Airfoil Development System," MIT, [Online]. Available: http://web.mit.edu/drela/Public/web/xfoil/. [Accessed 13 April 2023]. |
| [49] | P. Kämpf, "Why are highly cambered airfoils rarely used in propellers or wings?," 29 March 2020. [Online]. Available: https://aviation.stackexchange.com/questions/76690/why-are-highly-cambered-airfoils-rarely-used-in-propellers-or-wings. |
| [50] | Jane's Information Services, Jane's All the World's Aircraft, 1948-1952. |
| [51] | I. H. Abbot, A. E. Von Doenhoff and L. S. Stevens, Summary of Airfoil Data, 1945. |
| [52] | P. K. C. Rudolph, "High-Lift Systems on Commercial Subsonic Airlines," NASA, 1996. |