Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

Department of Aerodynamics, Energetics and Propulsion

TRABAJO FIN DE MÁSTER

Accelerating Lattice Boltzmann Methods using a deep learning approach

Author:
M. Daniel COSTERO VALERO

Professors:
Dr. Michael BAUERHEIM
Dr. Nicolas GOURDAIN

Abstract — This project is focused on the Lattice Boltzmann Method (LBM) which was revealed as an effective solver to compute low-Mach number flows because of its high-accuracy and low-cost advection scheme. The equations to be solved in LBM are discretized in space, time and velocity, choosing a few discrete velocities among the continuous velocity space. The accuracy of this method is outstanding for low Mach number flows but the numerical costs of the model only allow to compute solutions for $M < 0.3$. To overcome this problematic, Deep Learning techniques seem to be a good alternative, as, once trained, they can produce almost instantaneously the desired output. In this project, a model inspired by the Prandtl-Glauert transformation has been created. To train this network, two different databases containing more than 2000 different simulations of the flow around the RAE2822 airfoil have been created and validated. The model should learn the transformations from an incompressible low-Mach pressure field to a corresponding one at high-Mach number, accurately predicting this complex field for all the possible regimes. These transformations are to be applied to a very accurate pressure field calculated with LBM, thus extrapolating these good characteristics to any desired pressure field in the transonic regime almost instantaneously.

ISAE
10, avenue Édouard Belin
BP 54032
31055 Toulouse CEDEX 4

Acknowledgements

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to the supervisors of this project, M. Bauerheim and N. Gourdain, whose invaluable contributions have encouraged and motivated me throughout all the internship.

Furthermore, I would also like to acknowledge professor V. Chapin, whose help in creating the database has been crucial.

A special thanks goes to my teammates on the department Antonio and Ekhi, that have taught me countless things and motivated me in the realization of this project.

Finally, I really want to thank my family and friends who have constantly supported me in everything I do.

Contents

List of Figures

Introduction

Artificial Intelligence (AI) recently emerges in the engineering fields as a new approach to handle complex systems and elaborate physical models [22]. Deep learning [13] is one of those methods, based on a training/validation technique, which has shown outstanding results. For instance, a Go virtual player (one of the most difficult problem in AI) has been recently trained using a deep learning strategy and has won for the first time a world-class professional in a five-game match in 2016 [28]. Besides, the same idea has been used to create Alpha Zero, a chess engine that has been able to beat the best existing software after only four hours of self-learning [29].

In fluid mechanics, a breakthrough in numerical methods can be expected by using such a technique to develop complex physical models or enhance current numerical solvers [26]. For instance, some outstanding results have already been achieved, like the acceleration of the traditional CFD solvers achieved by Tompson et al. in [33], replacing the advection term calculation for a deep neural network. P. Baqué et al. [3] have also achieve very impressive results using Deep Learning techniques to optimize the aerodynamic shape of a car, reducing the time around a 20% and allowing to solve problems that can not be resolved using traditional techniques.

This project is focused on the Lattice Boltzmann Method (LBM) [31] [12] which was revealed as an effective solver to compute low-Mach number flows because of its high-accuracy and low-cost advection scheme [36]. Compared with Navier-Stokes solvers, the equations to be solved in LBM are discretized in time, space, and velocity, the latter requiring a specific model known as lattice where a few discrete velocities are chosen among the continuous velocity space [32]. Such a method yields effective computation with outstanding accuracy at low Mach numbers. However, the accuracy and numerical costs of the LBM for higher Mach numbers ($\mathrm{M} > 0.3$) is still a challenge, which requires new developments [20].

Therefore, this project intends to improve current LBM methods using a deep learning strategy. To do so, the idea is to use the classical weakly compressible 2D formulation available in the code Palabos [1], where the velocity space is discretized with a standard lattice 2DQ9, i.e. where 9 discrete velocities are used, to compute very accurate solutions for low-Mach number flows. This solution is to be introduced into a Deep Learning model that has learned the required transformations to produce the pressure field at high-Mach number from the corresponding low-Mach number one, in a similar way that the Prandtl-Glauert transformation [7]. However, this model should work for any subsonic flow, being able to calculate all the shock waves and detachments of the boundary layer that might appear in the flow. If the model is properly trained, it should be able to extrapolate the good characteristics of the LBM into high-Mach number flows almost instantaneously, producing very accurate solutions for all the range of subsonic velocities.

Among all the possible applications this idea could have applied to, the bi-dimensional transonic flows around super-critical airfoils, specifically the RAE2822, has been set as the main focus of this project, as it is a very active research field with lots of experimental and numerical data available [27] [19]. The main goal is to have very accurate solutions for the distribution of the pressure over the airfoil, leading to a more precise calculation of the aerodynamic coefficients.

The bibliography of the project can be separated into three different topics, each of them developed in different chapters. Firstly, the complexity of the transonic regime is introduced and different methods to obtain the aerodynamic forces over the airfoil are presented in Chapter 1. Then, an introduction to Artificial Intelligence, with special interest in Artificial and Convolutional Neural Networks, can be found in Chapter 2. Chapter 3 exposes the main principles of the Lattice Boltzmann Method, with an overview at the end expressing its strengths and weaknesses. The methodology followed in this work, as well as the obtained results, are exposed in Chapter 4, followed by the Conclusions of the project and the Future Work.

Chapter 1

Transonic flows

The aeronautical industry has always dreamt of doing faster planes that allow customers to travel in the quickest way possible to their destinations while being as efficient as possible in order to reduce the fuel consumption and the polluting emissions. Nevertheless, a physical limitation appears when the flow over the extrados of the airfoil, faster than the actual plane, reaches supersonic speed, as a shock wave is formed to return the flow to the surrounding conditions.

This comes with an important augmentation of the drag of the plane due to the pressure difference between both sides of the shock wave, that pushes the aircraft back. In addition, it can induce a detachment of the boundary layer that reduces drastically the lift of the wing and that may lead to a dangerous situation for the pilot. To avoid this situation to happen, planes usually fly at smaller speeds trying to reduce the potential risk. The understanding of the flow in these complex situations plays a key role in the research field and may lead to important improvements in the efficiency of the wing in these configurations.

Depending on the conditions around the airfoil, one can separate the flow in five different regimes as completely different physical phenomena occur in the fluid. If the Mach number, the ratio between the inflow velocity and the speed of sound, is lower than 0.3, the flow is considered as incompressible. This situation happens around light aircraft or cars. If the velocity is raised, the compressibility effects get more and more important and have to be taken into account, but no discontinuity is formed yet. This typically takes place between Mach numbers of 0.3 and 0.8 and is the typical flight regime of commercial planes. A flow is considered to be in the transonic regime if one part of the fluid moves faster than the speed of sound and the other part moves slower. These conditions vary with the altitude and the temperature of the fluid, but a typical range could be between Mach 0.8 and 1.2. The lower limit is fixed by the critical Mach, the speed at which one point of the flow gets the speed of sound. This creates shock waves in the airfoil that raises its drag and may detach the limit layer, reducing the lift. This is not a desirable condition and engineers try

Range	Description
$M < 0.3$	Incompressible subsonic Flow
$0.3 < M < 0.8$	Compressible subsonic Flow
$0.8 < M < 1.2$	Transonic Flow
$1.2 < M < 5.0$	Supersonic Flow
$5.0 < M$	Hypersonic Flow

Table 1.1: Different flow regimes

to raise this critical Mach as much as possible to avoid this situation to happen during the cruise flight while flying as fast as possible. If all the flow around the airfoil has a speed higher than the speed of sound, the regime is called supersonic. A typical range for the supersonic regime is between Mach 1.2 and 5, when the hypersonic regime begins. This is the usual regime of the combat planes and firearm bullets. It produces oblique shock waves in both the leading and trailing edges and the airfoils are usually diamond-shaped. A summary of the different regimes is shown in Table 1.1.

If the fluid around the airfoil is considered to be air, the Reynolds number is of the order of some millions, which means that the contribution of the viscosity is negligible with respect to the advection terms: the effect of the viscosity is only important in the boundary layer, close to the walls of the airfoil. Besides, the flow is considered as bi-dimensional around the airfoil, neglecting all the tri-dimensional effects, and only thin airfoils are considered, where the cord is much bigger than the thickness, as they are used in almost every aircraft nowadays.

Two different approaches can be considered when trying to solve the flow around an airfoil. On the one hand, one can try to numerically solve the discretized Navier-Stokes equations using different turbulence models depending on the computational power available (Section 1.2). On the other hand, potential methods can be used, leading to similarity rules that allow a fast calculation of the pressure field for different regimes from an incompressible simpler case (Section 1.1).

1.1 Potential Method

If a picture of a typical flow around a thin airfoil (thickness relative to the chord lower than 8%) at low velocity is observed, one can see that the streamlines are almost straight and the airfoil barely disturbs the flow (Figure 1.1). This intuition leads to a reformulation of the Navier-Stokes equations, writing the variables as a sum of its unperturbed value plus the perturbation induced by the airfoil: $\mathbf{V} = [V_\infty + u,\ v]$ where $u \sim v \sim \epsilon << V_\infty$.

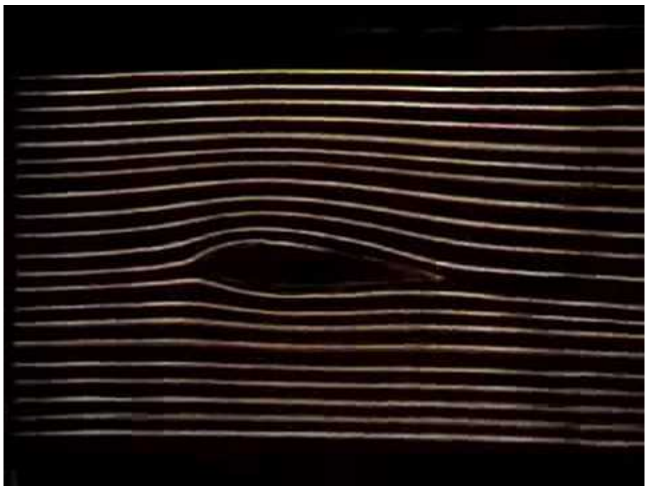

Figure 1.1: Streamlines of an airfoil at low speed

This hypothesis is not going to work in the leading edge, where the slope of the airfoil is too big, in detached flows where velocity perturbations are similar to the incident one or across shock waves where a discontinuity of velocity appears.

1.1.1 Incompressible flows

For flows with low velocity, typically with $M < 0.3$, the compressibility effects can be neglected and the Navier-Stokes equations get uncoupled. The mass conservation equation implies that the vertical perturbed velocity v has the same order of magnitude than the slope of the airfoil $\delta(x)$, and both have to be smaller than the incident velocity to fulfill the hypothesis of small perturbations.

If the flow is supposed irrotational, a potential of velocity, defined in Equation (1.1) can be used to simplify the Navier-Stokes equations. Φ corresponds to the total potential whereas ϕ refers only to the potential of the perturbation.

$$\begin{cases} \Phi = V_\infty \cdot x + \phi \\ grad(\phi) = u\vec{x} + v\vec{y} \end{cases} \tag{1.1}$$

Supposing inviscid, incompressible and stationary flow, the Navier-Stokes equations can be rewrote in terms of the potential of the flow as:

$$\Delta^2\Phi = \Delta^2\phi = 0 \tag{1.2}$$

This formulation reduces the number of variables to only one: the potential of the flow. Solving the flow around an airfoil is equivalent to solving the Laplace's equation with the corresponding limit and initial conditions. The critical condition is the one applied to the wall, which imposes that the flow must be tangent to the wall or, otherwise, that the component of the velocity normal to the airfoil must be zero (no flow penetrates the airfoil). This is expressed in Equation (1.3), where the surface of the wall is named $f^{\pm}(x)$ and its slope $\delta^{\pm}(x)$. The $+$ and - signs refer to the extrados and the intrados of the airfoil.

$$\begin{cases} (\vec{V} \cdot \vec{n})_{y=f^{\pm}(x)} = 0 \\ \vec{n} = \{-\delta^{\pm}(x)/|\vec{n}| \ , \ 1/|\vec{n}|\} \end{cases} \tag{1.3}$$

If one develops the equation and neglects the terms of order $u << V_{\infty}$, the following equation arises:

$$v(x,y) = V_{\infty} \cdot \delta^{\pm}(x) \tag{1.4}$$

As the airfoil is thin, v can be developed in a Taylor expansion around y=0 and, neglecting the terms of higher-order, one finds:

$$\boxed{v(x,0^{\pm}) = V_{\infty} \cdot \delta^{\pm}(x)} \tag{1.5}$$

Equation (1.5) provides the vertical perturbed velocity along the centerline of the airfoil as a function of the original geometry of the airfoil. As $\delta(x)$ is known from the geometry of the airfoil, v is also known. It can be deduced that the higher the curvature of the airfoil, the higher the vertical velocity. In the leading edge, its value is infinity and, in a flat plate, zero.

It is also important to calculate the pressure coefficient with these hypotheses, as it allows to calculate the pressure field:

$$\boxed{C_p(x,y) = \frac{p - p_{\infty}}{\frac{1}{2}\rho_{\infty} V_{\infty}^2} \approx -\frac{2u}{V_{\infty}}} \approx -\frac{2}{V_{\infty}} \cdot \frac{\partial \phi}{\partial x} \tag{1.6}$$

Knowing the distribution of vertical perturbed velocity $v(x)$, one has to solve the Laplace's equation (1.2) to calculate the field of the horizontal perturbed velocity $u(x,y)$, that provides the pressure field. The integration of this field gives the aerodynamics forces around the airfoil. This procedure is known as the direct problem (the inverse one would be calculating the shape of the airfoil from a given pressure field). The problem now has been simplified to the resolution of the Laplace's equation knowing $v(x)$. To do so, one of the most used methods is to assume a distribution of singularities (sources and vortexes) all along the central line of the airfoil. For further information, one can refer to [2] or [25].

1.1.2 Compressible subsonic flows

When the speed is raised, the effects of compressibility can not be neglected and the Navier-Stokes equations of mass and momentum conservation get coupled with the energy conservation one, increasing the difficulty of the problem. In addition, the density on each point of the flow must be calculated, adding one extra variable to the problem. To solve it, one extra equation is needed: the state equation of gases, usually considered as perfects.

The problem has now six equations (continuity, three of momentum, energy and state) for six variables (pressure, density, three velocities, and temperature): the problem is closed. These equations can be modified and expressed as functions of other variables, like the enthalpy, the entropy or the speed of sound, defined as:

$$a^2 = \frac{dp}{d\rho} = \gamma r T \tag{1.7}$$

With this definition and supposing that the flow is stationary, irrotational, of perfect fluid, without any external forces or heat flow and isoenergetic (same total energy in all the flow), the system of equations can be reduced to:

$$\begin{cases} a^2 \cdot div\vec{V} = \vec{V} \cdot \nabla(\frac{V^2}{2}) \\ h_t = h + \frac{V^2}{2} = Cst \\ S = Cst \\ p = \rho r T \end{cases} \tag{1.8}$$

This system of equations is valid for all the range of regimes, from zero speed up to hypersonic flows. It stops being valid when curved shock waves appear as they induce a rotation to the flow, that is not irrotational anymore. Following a similar reasoning as before, one can suppose that the perturbations induced by the airfoil are small and decompose the flow into the unperturbed flow and the perturbations, that will depend on the conditions of the incident flow and the geometry of the airfoil:

$$\vec{V}(M) = \vec{V_\infty} + \delta\vec{V} \quad with \;\; \delta\vec{V} << \vec{V_\infty} \tag{1.9}$$

As said before, this hypothesis is not valid close to the stagnation point or in detached flows or across shock waves. As the flow is irrotational, the velocity can be derived from a potential Φ defined in Equation (1.10). This potential can be decomposed into the incident flow potential and the perturbation potential ϕ. From the previous Equations (1.8), one can express:

$$\vec{V} = (V_\infty + u, v, w) = \vec{\nabla}\Phi = \vec{\nabla}(V_\infty \cdot x \;+\; \phi(x,y,z)) \tag{1.10}$$

$$\begin{aligned}(a^2 - (V_\infty + u)^2)\frac{\partial^2\phi}{\partial x^2} + (a^2 - v^2)\frac{\partial^2\phi}{\partial y^2} + (a^2 - w^2)\frac{\partial^2\phi}{\partial z^2} \\ - 2(V_\infty + u)\cdot v\frac{\partial^2\phi}{\partial x\partial y} - 2vw\frac{\partial^2\phi}{\partial y\partial z} - 2(V_\infty + u)\cdot w\frac{\partial^2\phi}{\partial x\partial z} = 0\end{aligned} \tag{1.11}$$

a in the previous equation makes reference to the local speed of sound, which depends on the local temperature. One can express this velocity as a function of the local velocity and the unperturbed conditions:

$$a^2 = a_\infty^2 - \frac{\gamma - 1}{2}(V^2 - V_\infty^2) = a_\infty^2 - \frac{\gamma - 1}{2}(2V_\infty u + u^2 + v^2 + w^2) \tag{1.12}$$

If the Equation (1.12) is introduced into Equation (1.11), the resulting equation only depends on the velocity and its derivatives. Keeping only the principal terms, the following equation can be written:

$$\begin{aligned} &(1 - M_\infty^2)\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = \\ &= (\gamma+1)M_\infty^2 \frac{u}{V_\infty}\frac{\partial^2\phi}{\partial x^2} + 2M_\infty^2 \frac{v}{V_\infty}\frac{\partial^2\phi}{\partial x\partial y} + 2M_\infty^2 \frac{w}{V_\infty}\frac{\partial^2\phi}{\partial x\partial z} + 2M_\infty^2 \frac{v}{V_\infty}\frac{w}{V_\infty}\frac{\partial^2\phi}{\partial y\partial z} \end{aligned} \tag{1.13}$$

This equation, valid for all the regimes, can be simplified assuming that all the second derivatives, except for $\frac{\partial^2\phi}{\partial x^2} = \frac{\partial u}{\partial x}$, have the same order. Doing so, one can find:

$$(1 - M_\infty^2)\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = (\gamma+1)M_\infty^2 \frac{u}{V_\infty}\frac{\partial^2\phi}{\partial x^2} \tag{1.14}$$

From this equation, putting a velocity close to zero, one can easily find the Laplace's equation, governing the incompressible flows (Equation (1.2)). For transonic flows with M close to 1, it can be shown that $\frac{\partial^2\phi}{\partial x^2}$ has to be of a superior order that the rest of the second derivatives [25]. For regimes completely subsonic or supersonic, the equation reduces to Equation (1.15), where the character of the equation changes from elliptic to hyperbolic for the two regimes. Imposing the limit conditions to these equations, one can find the pressure and velocity fields.

$$\boxed{(1 - M_\infty^2)\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = 0} \tag{1.15}$$

To calculate the pressure field, one can follow a similar procedure. Supposing that the flow is isentropic as was done before, the Equation (1.16) can be deduced, and, introducing the Equation (1.12) and keeping the first-order terms (as was done for the potential equation), one can find the Equation (1.17), that expresses the pressure coefficient distribution as a function of the perturbed velocity in the direction of $\vec{V_\infty}$.

$$\frac{p}{p_\infty} = \left(\frac{a}{a_\infty}\right)^{\frac{2\gamma}{\gamma-1}} \tag{1.16}$$

$$\boxed{C_p = \frac{p - p_\infty}{\frac{\gamma}{2}p_\infty M_\infty^2} = -\frac{2\cdot u}{V_\infty}} \tag{1.17}$$

1.1.3 Similarity rules

In both cases, the pressure field is obtained from the perturbation potential ϕ, which is calculated by solving the Equations (1.2) and (1.15). One can observe that they are very similar, only being different on a factor $\beta^2 = 1 - M_\infty^2$. Equation (1.2) is easier to solve and well-known solutions can be calculated supposing a distribution of sources and vortexes along the airfoil, as showed in [2] or [25]. Nevertheless, Equation (1.15) is harder to solve but, instead of doing it, one can calculate its solution using the solution of a corresponding incompressible flow of reference. This method is called the Prandtl-Glauert transformation, named after its creators, and the factor β also takes this same name.

Considering the same airfoil in compressible and incompressible flows $y = f(x)$, one can suppose that the potential of the compressible case is linked to the reference one by a linear relation (Equation (1.18)), where μ and η are two unknown constants that must be determined and the subscript $_0$ corresponds to the reference, incompressible case. This hypothesis is valid as only the linear terms have been kept in the development of the equation of the potential (Equation 1.15)).

$$\Phi(x, y) = V\ x + \phi(x, y) = \mu\ V_0\ x_0 + \eta\ \phi_0(x, y) \tag{1.18}$$

Let us suppose a coordinate change, like the one shown in Equation (1.19), that keeps the shape of the airfoil. The relationship between both regimes reduces to find the values of μ, η, A and B. To do so, some compatibility relations must be fulfilled.

$$\begin{cases} x = A\ x_0 \\ y = B\ y_0 \end{cases} \tag{1.19}$$

If the Equation (1.18) is introduced into the general equation for the potential (Equation (1.15)), one can find the Equation (1.20).

$$(1 - M_0^2)\frac{A^2}{B^2}\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{1.20}$$

As ϕ is a perturbation potential, it must satisfy the Equation (1.15), which provides the **first similarity rule**:

$$\boxed{(1 - M_0^2)\frac{A^2}{B^2} = (1 - M^2) \Rightarrow \frac{B^2}{A^2} = \frac{1 - M_0^2}{1 - M^2}} \tag{1.21}$$

As explained before, the condition of the flow tangent to the walls could be simplified to the simple Equation (1.5). Applying all the transformations, the **second similarity rule** can be found:

$$\boxed{\delta = \frac{df}{dx} = \frac{\eta}{\mu}\frac{A}{B}\ \delta_0} \tag{1.22}$$

Similar procedure can be made for the pressure coefficient (Equation (1.17)), as well as for the lift coefficient C_l, the drag coefficient C_d and the momentum coefficient C_m:

$$\boxed{\frac{C_p}{C_{p,0}} = \frac{C_l}{C_{l,0}} = \frac{C_d}{C_{d,0}} = \frac{C_m}{C_{m,0}} = \frac{\eta}{\mu}} \tag{1.23}$$

In the considered case, the shape of the airfoil must be the same both in incompressible and compressible regime. Imposing this condition in the Equation (1.22), one can find the relation $A/B = \mu/\eta$. From Equation(1.21), another relation for A/B can be easily found, finding the following relation that relates the aerodynamics coefficients with the Mach number of both regimes:

$$\boxed{\boxed{\frac{C_p}{C_{p,0}} = \frac{C_l}{C_{l,0}} = \frac{C_d}{C_{d,0}} = \frac{C_m}{C_{m,0}} = \sqrt{\frac{|1-M_0^2|}{|1-M^2|}}}} \tag{1.24}$$

This relation is valid for all the range of incident Mach number, from 0 to 5, where the hypersonic regime begins. The absolute value allows the squared root to be always positive, both in subsonic as in supersonic. The main problem of this relation is that it presents a singularity in $M = 1$, as showed in Figure 1.2. In addition, because of the hypothesis taken to get to this relation, it is not valid in places where the perturbations of the velocity are of the same order as the velocity itself. This situation happens in the leading edge, in detached flows or in shock waves. This method does not work properly for Mach numbers bigger than 0.7 as the hypothesis of small perturbations is less and less accurate. In order to improve the precision for higher velocities, the Karman-Tsien relation can be used (Equation (1.25)), which is based on the nonlinear hodograph solution that uses a linear approximation to the pressure-density relation instead of the isentropic one [35]. The Karman-Tsien relation can be applied locally to each point of the flow but it can not be extended to the rest of aerodynamic coefficients. However, they can be calculated by integration of the pressure field.

$$C_p = \frac{C_{p,0}}{\sqrt{1-M_\infty^2} + \frac{C_{p,0}}{2} \cdot \left(\frac{M_\infty^2}{1+\sqrt{1-M_\infty^2}}\right)} \tag{1.25}$$

In Figure 1.3, taken from [35], three different models to predict the compressibility of a flow are shown as well as the experimental data. It can be seen how the Karman-Tsien relation provides better results for Mach closer to 1 than those of the Prandtl-Glauert. This correlation is the one that is typically used for the calculation and is implemented in codes such as XFOIL.

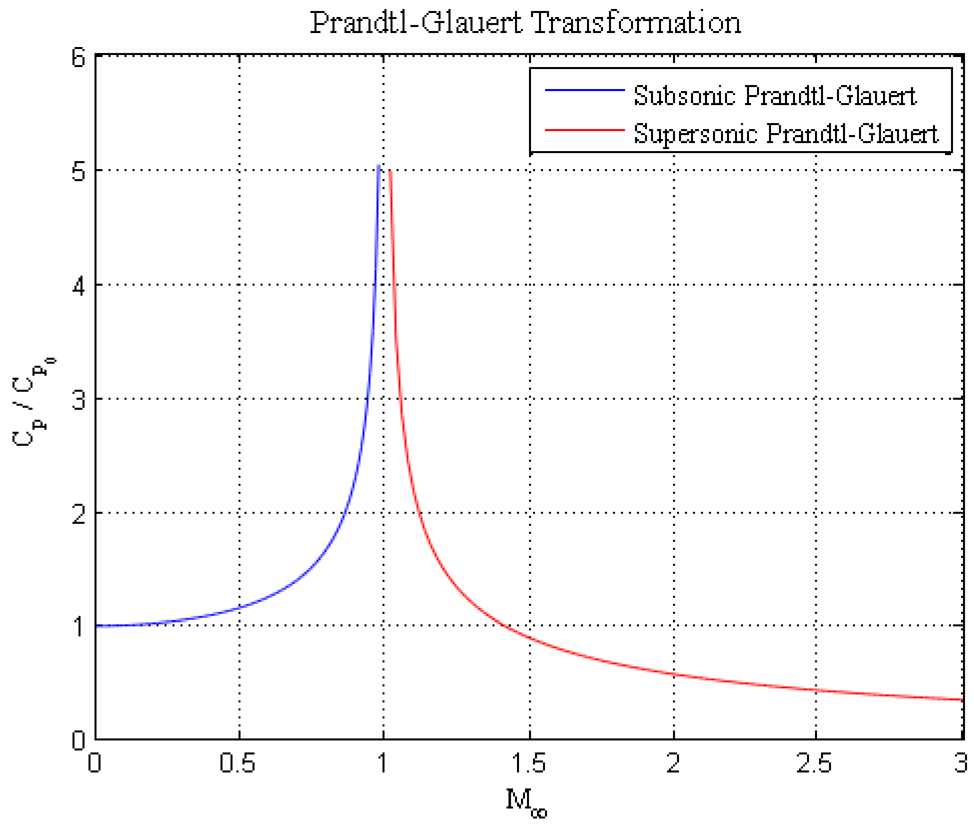

Figure 1.2: Prandtl-Glauert relation as a function of the Mach number, considering $M_0 = 0$

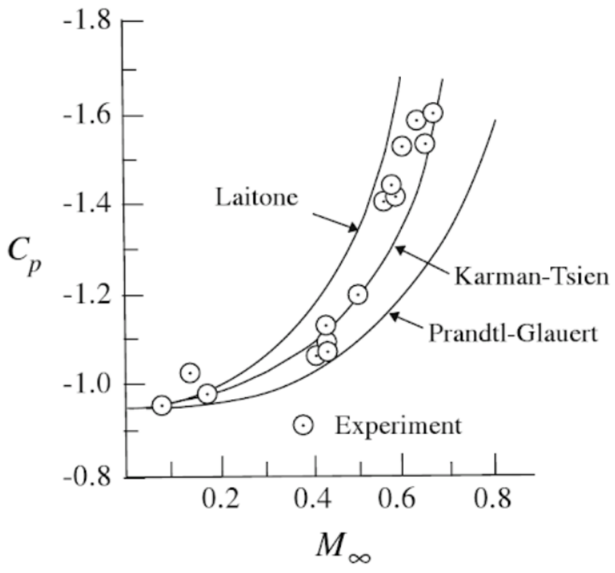

Figure 1.3: Three compressibility correction models compared to the experimental data obtained from a NACA 4412 at small angle of attack ($\alpha = 1.88^o$)

1.2 Numerical Methods

The potential methods provide accurate and fast solutions when the Mach number is not too high, up to values of $M < 0.6$, where the velocity perturbations induced by the airfoil are smaller than the incident velocity and the linearity hypothesis is still accurate. When the velocity rises above this value, some complex, non-linear phenomena start to appear, such as shock waves, detachment of the boundary layer or re-circulation bubbles, and the simple potential method can not describe them.

To solve these cases, the most common procedure is to solve the Navier-Stokes equations numerically, discretizing spatially the fluid region around the airfoil. These methods are able to simulate correctly these phenomena and, in addition, do not present the divergence problem on the stagnation point.

The problem of these methods arises in the election of the size of the used mesh. To correctly and completely model the turbulent flow, cells of the size of the Kolmogorov scale η [11], where viscosity dominates and the turbulent kinetic energy is dissipated into heat, are needed. This scale is calculated in Equation (1.26) and it is proportional to the inverse of the Reynolds number of the incident flow [23]. To get an idea, if a typical Reynolds number of 5 million is taken over an airfoil with a chord of 1 meter, the Kolmogorov's scale is around $10\mu m$. This means that 100.000 cells are needed along the airfoil. If a typical squared domain of 20 chords in both directions is considered, it will have a total of $4x10^{12}$ cells that have to be solved. This is similar to the number of trees on Earth or the number of cells in the human body. In conclusion, is practically impossible to solve the flow around an airfoil using this method, known as DNS (Direct Numerical Simulation) [18].

$$\eta \sim Re_L^{-3/4} \cdot L \tag{1.26}$$

Instead of solving the Navier Stokes equations completely, some turbulence models allow to model the smallest scales thus permitting the use of thicker meshes that take reasonable amounts of computing time and memory to be solved. However, they must be thin enough to capture the discontinuities that may appear in the flow.

The Large Eddy Simulation (LES) for example, uses a low-pass filter to simplify the Navier Stokes equations, ignoring the low scale phenomena as, following the Kolmogorov's similarity rule, is universal an independent of the considered problem [23]. Using LES, the large eddies are solved completely and a subgrid-scale model (SGS model) is used to solve the smallest scales. These models are very precise but they still require a lot of computation time and memory. They are currently being used to solve combustion and acoustic problems for example.

The most common way to solve turbulent problems is using the Reynolds-Averaged Navier-Stokes (RANS) equations, which are a time-averaged version of the classical equations. This method divides the values of the magnitudes between its average and its oscillating parts.

$$u(\mathbf{x},t) = \overline{u(\mathbf{x})} + u'(\mathbf{x},t) \tag{1.27}$$

Introducing this decomposition in the original equations and taking into account that $\overline{u'(\mathbf{x})} = 0$, the Equation (1.28) can be obtained. The last term of this equation is known as the Reynolds stress tensor and represents the contribution of the turbulence to the problem. Its value is not known and it needs to be modeled. Depending on the way of modelling this term, several methods have been developed.

$$\frac{\partial(\rho\overline{u_i})}{\partial t} + \frac{\partial(\rho\overline{u_i u_j})}{\partial x_j} = -\frac{\partial\overline{p}}{\partial x_i} + \frac{\partial\overline{\tau_{ij}}}{\partial x_j} + \overline{\rho f_i} - \frac{\partial(\rho\overline{u'_i u'_j})}{\partial x_j} \tag{1.28}$$

Most of them are based on the Boussinesq hypothesis that, inspired by the constitutive equation of the solid materials [15], proposed the Equation (1.29), where δ is the Kronecker delta. It introduces a new variable called the turbulent viscosity $\nu_t(\mathbf{x},t)$ and the turbulent kinetic energy defined as $k = 1/2 \cdot \overline{u'_i u'_i}$.

$$-\overline{u'_i u'_j} = \nu_t \cdot \left(\frac{\partial\overline{u_i}}{\partial x_j} + \frac{\partial\overline{u_j}}{\partial x_i}\right) - 2/3 \cdot k \cdot \delta_{i,j} \tag{1.29}$$

In order to close the problem, a model for ν_t is needed. One of the simplest models is the Spalart–Allmaras [30], which solves a transport equation for the turbulent viscosity introducing some numerical constants tuned from experimental results. It works well in aerospace problems and in problems with flows around walls. Some additional modifications have been added to the original formulation to improve the precision in regions close to the wall, with a so-called Enhanced Wall Treatment.

The most used method is the $k - \epsilon$ [5], that models the turbulent viscosity as $\nu_t = C_\mu \cdot k^2/\epsilon$, where ϵ is the rate of dissipation of turbulent energy. In this case, two transport equations are solved: one for k and one for ϵ. The fact of introducing two additional equations raises the computational capabilities required to perform this type of simulations, but the precision achieved, especially in shear layers flows, is considerably bigger than the one provided by simpler methods.

Following a similar philosophy appears the $k - \omega$ model, which introduces the specific rate of dissipation of the turbulent kinetic energy k into internal thermal energy ω. In this case, $\nu_t = k/\omega$ and the transport equations for k and ω are solved. This model provides better results in the boundary layer than the $k - \epsilon$ and is usually used in aerodynamics.

In addition to all these methods based on the Navier-Stokes equations, there also exists the possibility of solving the Boltzmann equation, based on a mesoscopic scale of the physic. This method is known as the Lattice Boltzmann method and is explained later in the Chapter 3.

Chapter 2

Artificial Intelligence

2.1 What is Artificial Intelligence?

Artificial intelligence (AI) is a very extensive concept whose definition is not completely clear as it covers a lot of very different tasks. According to Andrew Moore, Former-Dean of the School of Computer Science at Carnegie Mellon University, “Artificial intelligence is the science and engineering of making computers behave in ways that, until recently, we thought required human intelligence” [10]. This definition implies that the Artificial Intelligence evolves with time and what used to be considered as AI 20 years ago is no longer considered as such.

The most famous example of this variability is the famous IBM’s chess machine Deep Blue, which was able to win a match against the world chess champion Garry Kaspárov in 1997. This machine uses a method called tree search algorithms to evaluate millions of moves at every turn, getting to decide which one is the best[4]. Nowadays, developments of Deep Blue are available on any computer and creating a program to play chess is no longer consider as IA because it is seen as a calculator of different variations rather than a program thinking as a human mind.

Machine learning (ML) is a subset of artificial intelligence, and as defined by Computer Scientist and machine learning pioneer Tom M. Mitchell: “Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience” [10]. This is the kind of algorithm used by companies such as Spotify, Amazon, YouTube or Netflix, where they are able to suggest to the user some products based on the previous experience of the customer on their websites. The algorithm learns the preferences and tastes of the user from previous interactions and is able to pick, among all the available products, the ones that are going to maximize the revenue of the company.

Among all the techniques and algorithms classified as ML, **Deep Learning (DL)** is a subset of them that deals with very complex problems that require more advanced

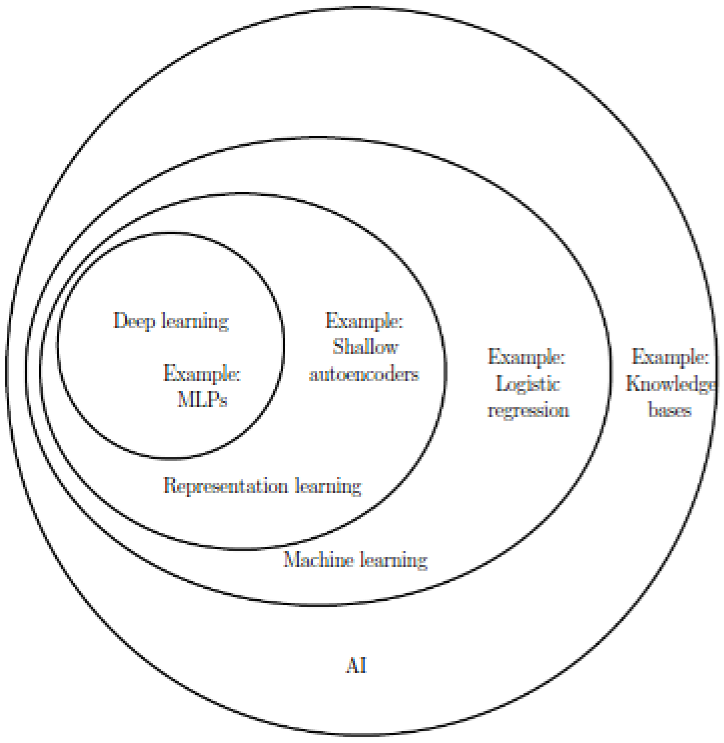

Figure 2.1: Classification of the different machine learning techniques

and specific algorithms than traditional AI problems. DL is used in speech and audio recognition, computer vision or natural language processing for example. Deep learning solutions are based on Artificial Neural Networks (ANN) and are the main field of research of machine learning scientists. A scheme summarizing the relation between IA, ML and DL is shown in Figure 2.1, taken from [8], the reference book for Deep Learning.

2.2 Machine Learning

Machine Learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. A more formal definition is given again by Tom M. Mitchell by: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"[17].

All the machine learning techniques can be classified into two different groups: supervised and unsupervised learning.

- **Supervised learning.** In these types of techniques, some input data and their correspondent desired outputs are given to the algorithm. With all these data, the algorithm learns by itself the different relationships between the inputs and outputs, creating an internal model. This model is able to “predict” the output for any given example. An example of supervised learning would be the prediction of the price of a house given its size and its position and a collection of examples of the adjacent houses.
- **Unsupervised learning.** In these types of techniques, only a list of data is given, without any label or desired output. The set of algorithms are able to find by itself some inner patterns and structures. An example of unsupervised learning is the suggestions of webs such as Amazon or YouTube. They know what people do in their webs: what they search, what they look, where they click, etc., and with all these data, they are able to predict your interests and use it with commercial purposes.

Machine learning algorithms are usually used for two main purposes: classification and regression. In classification problems, a label is given to each point of the data, both in supervised and unsupervised learning. A typical example of that is the spam filtering, classifying emails as “spam” or “not spam”. In regression problems, the prediction is made over continuous variables, such as the prize of a house or the power consumption of a building.

In Machine Learning, the data is usually divided into three different sets: training set, cross-validation set and test set. A typical separation would be around 60%, 20% and 20% of the total set of data, respectively, but it is very dependent on the considered program. The train set is used by the algorithm to find the inner relationships in the data and create a model. The cross-validation set is used to tune the parameters of the algorithm and optimize the model. The data of the training set is trained with different parameters of the algorithm and its performance is analyzed in the cross-validation data, choosing the set of parameters that reduces the error predicting the cross-validation. Once the model is optimized, the test set is used to evaluate the accuracy of the predictions made by the model.

There are several algorithms used in supervised machine learning, such as artificial neural networks, decision trees, support vector machines or genetic algorithms. All of them have their advantages and disadvantages, being the election of the correct algorithm a key decision to successfully solve the problem.

Artificial Neural Networks (ANN)

ANNs are algorithms inspired in the functioning of the brain where there are a lot of ’neurons’ connected to each other, exchanging information between them and creating a complex network. A typical neuron receives information from some other neurons and evaluates a function with a linear combination of the different received information, returning a real value that is going to be used for other neurons. It is a supervised learning algorithm that 'learns' by calibrating the inner parameters of the network. A typical diagram of this type of network is shown in Figure 2.6 and a detailed explanation of this method can be found in Section 2.3.

Support Vector Machine (SVM)

SVM is a supervised algorithm used to classify data into two different types. It creates a boundary between the two types of data, maximizing the distance of the points to this boundary. By doing that, two regions of the space of variables are created and the algorithm predicts the label of any new example depending on which region of the space the sample is placed. A diagram of how a Support Vector Machine works is showed in Figure 2.2. As it is shown, line H1 is not able to separate correctly both labels. Even if H2 and H3 create a correct boundary, it is clear that the reliability of line H3 is bigger than the one of H2, and it is the one that is calculated by SVM.

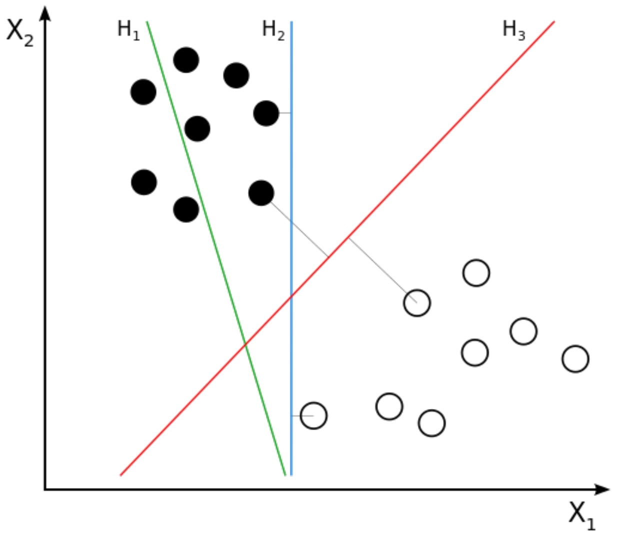

Figure 2.2: Support Vector Machine

Genetic algorithms (GA)

GAs can also be used in machine learning. Specifically, a cost function is defined as the difference between the model to be created and the training data and needs to be minimized. As it depends on a large number of different variables, easily several million,

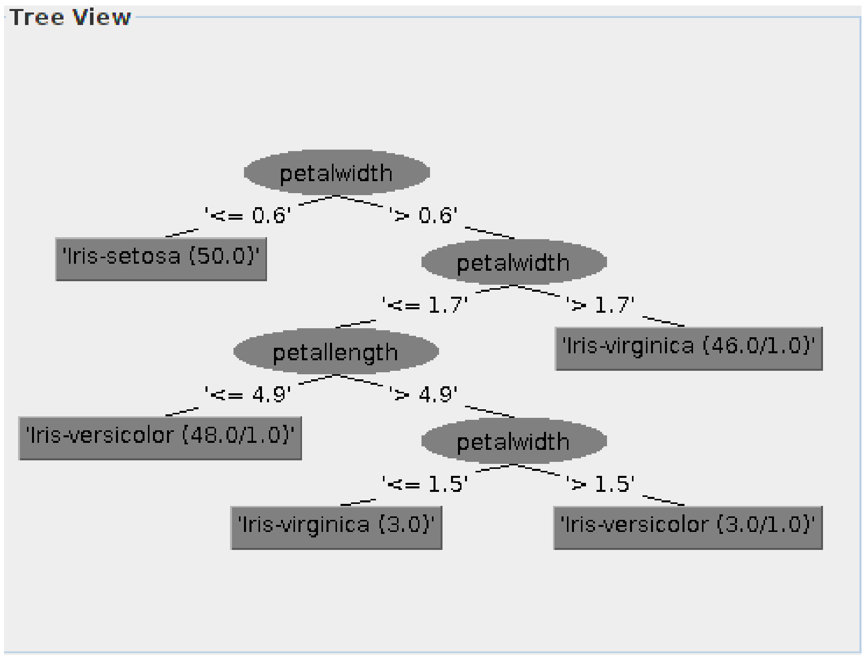

Figure 2.3: Decision tree diagram

traditional minimization techniques may be taken too long and genetic algorithms can be used to reduce the calculation time.

Decision Trees

The goal of this family of algorithms is the same as any other machine learning techniques: create a model able to predict an output from a set of input variables. If the outputs are different labels, then it is a classification tree; if it is one (or more) continuous real number, it is called a regression tree. To create this model, one question about a single variable is asked on each node: is variable X bigger than a value x? According to the answer, two different leaves are created. Sometimes, more than two leaves can be created, but it provides similar results than asking two consecutive questions. This question is chosen internally by the tree, following different statistic techniques. On each node, a lot of different questions are tried and the one that provides most information (reduces the 'entropy' of the data) is chosen. The criterion to determinate which question is selected is usually the one that minimizes the root mean squared error or the mean absolute error of the answer. Sometimes, some other criteria are followed to separate concrete points of the data with more specific questions that do not minimize the error for this specific question but they do optimize the global performance. An example of the decision process made by a tree is showed in Figure 2.3.

A single tree is able to separate correctly all the different data up to the point where only a single point is inside each node. This model is clearly overfitting and will not be able to predict correctly. To avoid this situation, some stopping criterion needs to be introduced. The problem is that this criterion is not easy to choose because the path followed by the tree is unknown. A similar procedure to the one explained before, using cross-validation, could be used, but the change of the parameters can affect the way in which the splits are made, obtaining a total different tree for each set of parameters. To correct this problem, several different trees are built and all the results are processed in order to choose the final one. One of these methods is called Random Forest and is explained below.

Random Forest

To solve the overfitting problem of single decision trees, a lot of different techniques have been developed. One of the most used is called Random Forest and is based on the idea of building several trees, all of them different from the others, to solve the same problem. As each tree is unique, the predicted values are going to be different. The final output is chosen to be an average of all the outputs in regression problems or the mode for the classification ones.

To build different trees, different data sets are needed. To create this subsets of data, a technique called Bagging (or **B**ootstrap **agg**regat**ing**) is used. It divides the data set (N samples) into m subsets of n samples, by randomly sampling with replacement. If n=N, approximately 67% of the data is going to be original, and the rest is going to be copies of those ones. Even with m different trees, if some variable is very important for one tree, it is going to be important for all the other trees and the resulting forests are going to be very similar one to another. In addition, evaluating all the variables on the split of each node takes a lot of time and resources. This problem is solved with a technique called Feature Bagging. On each node, only a randomly chosen subset of the variables is analyzed and the one that provides the most relevant information is chosen. This number is usually taken as $\sqrt{M}$, where M is the number of attributes or variables of the problem.

2.2.1 Fitting problems

The main objective of any machine learning algorithm is to create a model from a set of given data points able to accurately predict new output values from new data. The capacity of extrapolation is the most desired capacity of these algorithms and the parameters of the algorithm must be tuned correctly to achieve it.

Every machine learning algorithm has a set of parameters that must be optimized to minimize a determined cost function, whose election is a crucial decision for the model to work properly. A typical cost function (Equation (2.1)) measures the total average distance between the created model and the data set. If the value of the cost function is too big,

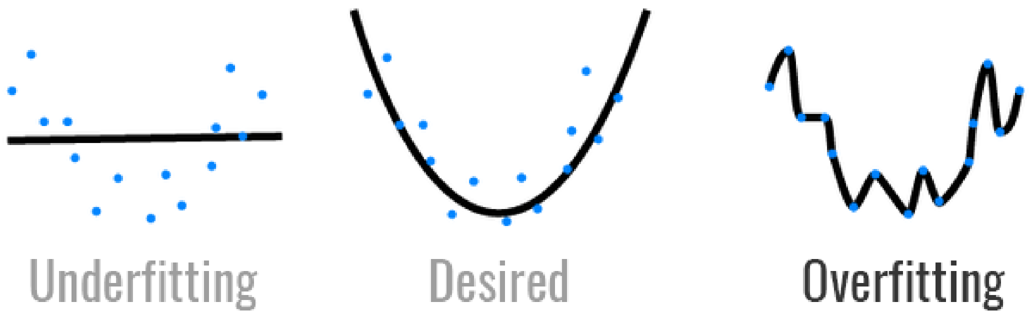

Figure 2.4: Fitting problems

the model does not fit the data correctly and the created model is not able to predict any new value. An example of this situation, called **underfitting**, is showed on the left side of Figure 2.4. To correct this type of error, some parameters of the model need to be changed or some extra data need to be provided to the algorithm.

$$J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{2.1}$$

On the opposite side, if the value of the cost function is too low, the model follows completely the data and loses its predictive capacity. This situation is called **overfitting** and is a very typical problem of every Machine Learning model (right side of Figure 2.4). The best way to reduce this problem is reducing the number of parameters of the algorithm, making it simpler. The optimal situation is between the two previous ones, where the model only follows the trend of the data but is still able to predict correctly new values for the desired output.

Many different techniques exist to avoid overfitting. All of them are based on adding restrictions to the inner parameters of the algorithms. In ANN for example, the typical cost function is modified (Equation (2.2)) adding some extra parameters λ:

$$J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \cdot \theta \tag{2.2}$$

$x^{(i)}$ and $y^{(i)}$ are each one of the training examples, h represents the created model as a function of θ, the set of original parameters of the algorithm, and λ is the newly created set of parameters. λ has usually the same value for all the parameters, but it can be imposed individually. If a large value is given to λ, the optimization of the cost function is going to find small values for θ, trying to make the second part of the equation as low as possible. By doing so, some parameters will be set to very low values (or even zero), simplifying the complexity of the model and reducing the overfitting.

It is important to mention that the choice of a correct value for λ is critical to avoid overfitting. If a small value is taken, few parameters are going to be removed and the model will still be overfitting. On the other hand, if a big value is set, the second term of the equation is going to be more important than the original cost function and the model is going to underfit. To select the correct value of λ, the cross-validation data set is used. Taking only the training set of the data, several models are trained using different values of λ. These models are used to predict values from the cross-validation set, from which the output is known. The model whose predictions are closer to the values of the cross-validation set provides the optimum value of λ.

2.2.2 Metrics

All machine learning algorithms, once they have been trained, are able to predict outputs for a given set of new data. In order to measure how good thess predictions are some metrics need to be defined. In machine learning, three metrics are the most widely used: MAE, RMSE and R^2.

- **Mean Absolute Error (MAE):** is defined in Equation (2.3). It measures the difference between the model prediction y_i and the real value x_i of example i.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{2.3}$$

- **Root Mean Squared Error (RMSE):** is defined in Equation (2.4). The behavior is similar to the one in MAE, but averages the square of the difference. This metric gives more importance to points with a high deviation, raising its value.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}} \tag{2.4}$$

  Both MAE and RMSE are absolute metrics and give information about the global error. Sometimes, relative measures of these parameters are used, comparing them with the maximum value of the measures.

- **Coefficient of determination (R^2):** is defined in Equation (2.5).

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{2.5}$$

  Where:

$$SS_{tot} = \sum_i (y_i - \overline{y})^2 \tag{2.6}$$

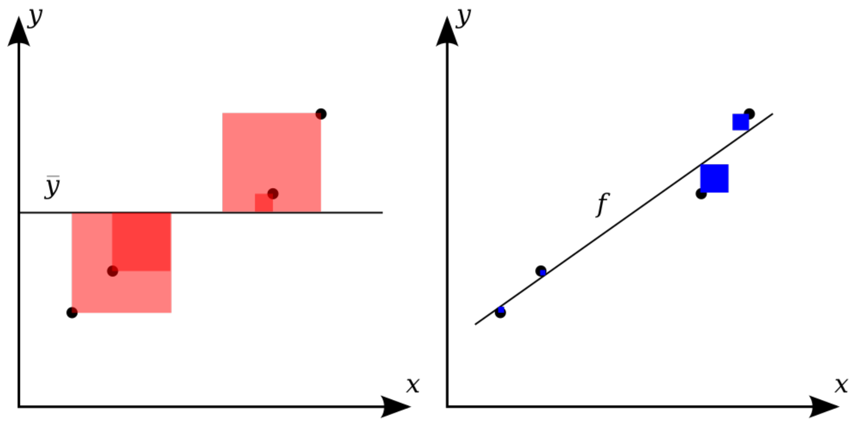

Figure 2.5: Coefficient of Determination

$$SS_{res} = \sum_i (y_i - h_i)^2 \tag{2.7}$$

$\overline{y}$ is the average of all the data, y_i are the real values and h_i are the predicted values. In summary, R^2 compares how good is the model (blue area on the right side of Figure 2.5) in relation to a model consisting of a horizontal line with the mean value of the considered data (red area on the left side of Figure 2.5).

A model that fits perfectly the data would have a value of 1 for the coefficient of determination. If it fits as well (or bad) as the horizontal line, it would have a zero value. It could even have a negative value if the model is worst that the horizontal line.

2.3 Artificial Neural Networks (ANN)

Artificial Neural Networks are the most widely used method in Machine Learning. A typical network is composed of different layers, as shown in Figure 2.6, being the first one the 'input layer' and the last one the 'output layer'. In between, it can be any number of layers, depending on the complexity of the problem. Each layer has a determined number of independent neurons. The number of neurons in the input layer is fixed by the number of variables of the data and the size of the output is fixed by the number of variables to be predicted. The number of hidden layers and its number of neurons are hyperparameters of the model and must be chosen wisely by the user.

Each neuron of a given layer is connected to all the neurons of the previous and the following layers but it is independent of the neurons of its same layer. Each neuron has assigned one activation value, which is expressed as $a_i^{(l)}$, where l makes reference to the

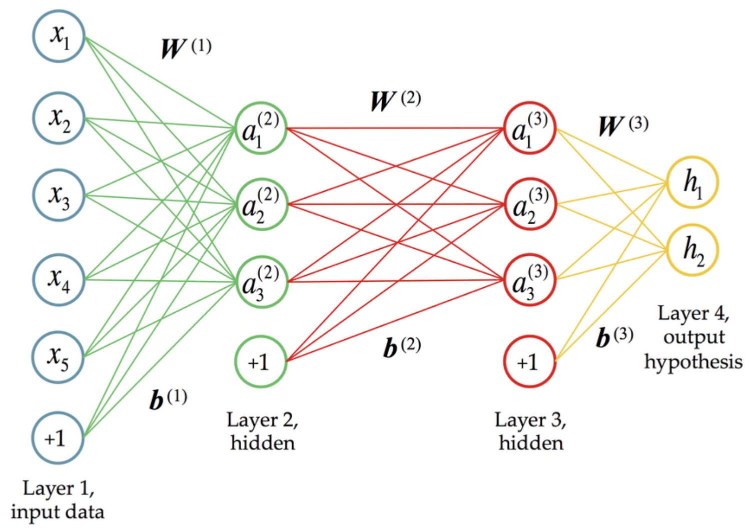

Figure 2.6: Artificial Neural Network

number of the hidden layer and i refers to the number of the neuron of the layer l. Each connection also has a so-called 'weight' assigned, creating a matrix between each layer expressed as $\overline{W^{(l)}}$, where l refers to the connections between the layer $l-1$ and l. Each component of the matrix refers to a single connection between neurons, being $w_{i,j}^{(l)}$ the connection between the neuron i of the layer $l-1$ and the neuron j of the layer l. The value entering the neuron $z_i^{(l)}$ is calculated as a linear combination of the neurons of the previous layer:

$$z_j^{(l)} = \sum_i w_{i,j}^{(l)} \cdot a_i^{(l-1)} + b_j^{(l)} \tag{2.8}$$

The $\overline{b}$ that appears in the previous equations is called the bias vector and represents the connections between an extra neuron of unitary value added to the layer that is used to improve the accuracy of the model, adding independent terms. In the end, this is similar to using a linear model like $z = ax + by + c$, where a, b represent the weights, x, y the activation values and c the bias term.

Repeating the same procedure for all the neurons, the output values will be a linear combination of the values of all the neurons and the model will not be able to learn the non-linear effects. To correct that, some non-linearities can be introduced in the model, calculating the activation values of the neurons a as a non-linear function of its 'input' value z. Like that, the activation values can be expressed as:

$$a_j^{(l)} = \sigma^{(l)} \left(\sum_i w_{i,j}^{(l)} \cdot a_i + b_j^{(l)} \right) \Rightarrow \overline{a^{(l)}} = \sigma^{(l)} \left(\overline{W^{(l)}} \cdot \overline{a^{(l-1)}} + \overline{b}^{(l)} \right) \tag{2.9}$$

where $\sigma^{(l)}$, called the activation function, represents the non-linear function of the layer

l. Different functions can be used for each of the layers and they affect enormously the performance of the network. A further explanation about the activation functions can be found in section 2.3.2.

Once an architecture of the network and the activation functions have been chosen, the output of the model is only a function of the weights of the connections. So the problem now is reduced to find the values of these parameters that optimize the performance of the model. To do so, one needs to define a way to measure how good the model is using a cost function. Different cost functions are used depending on the considered problem and its definition is one of the most important decisions to be taken in the design of the model.

In classification problems where the output of the model is a 1 or a 0, the cost function is the accuracy of the model: the number of examples classified correctly over the total number of examples. In regression problems, this choice is not so obvious as the output of the model is a real value that is going to be more or less close to the desired value. Typical choices are the RMSE or the MAE functions, explained in section 2.2.2, but more complex functions can be used.

Once the cost function is defined, the problem reduces to find the values of the weights that minimize the cost function. To do so, the gradient descent method is used. Firstly, the weights are randomly initialized and, after each iteration, their values are changed in the direction of the gradient, following the Equation (2.10).

$$\Delta w_{i,j}^{(l)} = -\alpha \cdot \frac{\partial J}{\partial w_{i,j}^{(l)}} \tag{2.10}$$

α is the learning rate and represents the size of the steps taken on each iteration. If it is very small, a lot of iterations are needed to find the optimum solution. On the contrary, setting a value too big may lead the algorithm to diverge as it may jump over the minimum. The typical value is around 0.001 but it must be tested and optimized for each problem.

The problem lies now on finding the gradient of the cost function in relation to each single weight of the network. To do so, the backpropagation algorithm is used, simplifying enormously the task.

2.3.1 Backpropagation

The backpropagation algorithm is based on the chain rule, using the gradients in a layer to compute those on the following one. If an input is given to the network, it is going to pass through all the layers providing an output on the last layer L named as $\overline{a^{(L)}}$. With this value, one can compute the value of the cost function J and all the weights of the network are known. The next step is to calculate the gradient of the cost function with respect to the last layer weights and bias applying the chain rule:

$$\frac{\partial J}{\partial w_{i,j}^{(L)}} = \frac{\partial J}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial z_i^{(L)}}{\partial w_{i,j}^{(L)}} \tag{2.11}$$

$$\frac{\partial J}{\partial b_i^{(L)}} = \frac{\partial J}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial z_i^{(L)}}{\partial b_i^{(L)}} \tag{2.12}$$

The first term of both equations is the derivative of the cost function with respect to the predictions of the network. It is a simple function defined by the user whose analytical derivative is easy to calculate. The second term is the derivative of the activation function and is also known analytically. These two terms multiplied represent the responsibility of the neuron to cause the error of the model and are usually written as $\delta^{(L)}$. The last term of both equations is also very easy to calculate:

$$\frac{\partial z_i^{(L)}}{\partial w_{i,j}^{(L)}} = a_i^{(L-1)} \tag{2.13}$$

$$\frac{\partial z_i^{(L)}}{\partial b_{i,j}^{(L)}} = 1 \tag{2.14}$$

Multiplying all of these values, the gradients with respect to the last layer are calculated like:

$$\frac{\partial J}{\partial w_{i,j}^{(L)}} = \delta^{(L)} \cdot a_i^{(L-1)} \tag{2.15}$$

$$\frac{\partial J}{\partial b_i^{(L)}} = a_i^{(L-1)} \tag{2.16}$$

To calculate the gradients of the previous layer $L-1$, a similar procedure is used:

$$\frac{\partial J}{\partial w_{i,j}^{(L-1)}} = \frac{\partial J}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial z_i^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \cdot \frac{\partial z_i^{(L-1)}}{\partial w_{i,j}^{(L-1)}} \tag{2.17}$$

$$\frac{\partial J}{\partial b_i^{(L-1)}} = \frac{\partial J}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial z_i^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \cdot \frac{\partial z_i^{(L-1)}}{\partial b_i^{(L-1)}} \tag{2.18}$$

Here, the two first terms are equal to $\delta^{(L)}$, calculated in the previous layer. The third term is, by definition, the matrix with the weights of the L layer $\overline{W^{(L)}}$. The forth term is the derivative of the activation function of the $L-1$ layer, known analytically, an the last term are just the activation values of the neurons of the previous layer $L-2$ for the derivative with respect to the weights and 1 for the ones with respect to the bias term. This results in:

$$\frac{\partial J}{\partial w_{i,j}^{(L-1)}} = \delta^{(L)} \cdot w_{i,j}^{(L)} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \cdot a_i^{(L-2)} \tag{2.19}$$

$$\frac{\partial J}{\partial b_i^{(L-1)}} = \delta^{(L)} \cdot w_{i,j}^{(L)} \cdot \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \tag{2.20}$$

This procedure can be followed along all the depth of the network, getting to calculate the partial derivatives of the cost function with respect to all the weights of the network. These derivatives can be used in the gradient descent algorithm to change the values of all the weights of the network in this direction and minimizing the cost function at each iteration.

The gradients calculated using the backpropagation algorithm depend on the training example provided to the network because it is propagated forward to calculate all the weights and activations of the network. However, if only one example is used to calculate these gradients, the weights are going to change trying to approximate the model to this concrete example instead of having a global vision of all the training examples. To avoid this situation, the gradients are calculated averaging the values calculated for several different cases.

2.3.2 Activation functions

Unit activation function

The activation functions are responsible for the non-linearity of the model. Without them, only a linear model can be created, as proved in Equation (2.23), and the fact of having several layers would be useless as the network would work in the exact same way that a simpler one-layer model. The fact of not having an activation function can be seen as having a 'unit activation function' that returns the input value: $\sigma(x) = x$. Its behavior can be expressed as:

$$\overline{a^{(l-1)}} = \sigma^{(l-1)}\left(\overline{W^{(l-1)}} \cdot \overline{a^{(l-2)}} + \overline{b}^{(l-1)}\right) = \overline{W^{(l-1)}} \cdot \overline{a^{(l-2)}} + \overline{b}^{(l-1)} \tag{2.21}$$

$$\overline{a^{(l)}} = \sigma^{(l)}\left(\overline{W^{(l)}} \cdot \overline{a^{(l-1)}} + \overline{b}^{(l)}\right) = \overline{W^{(l)}} \cdot \overline{a^{(l-1)}} + \overline{b}^{(l)} \tag{2.22}$$

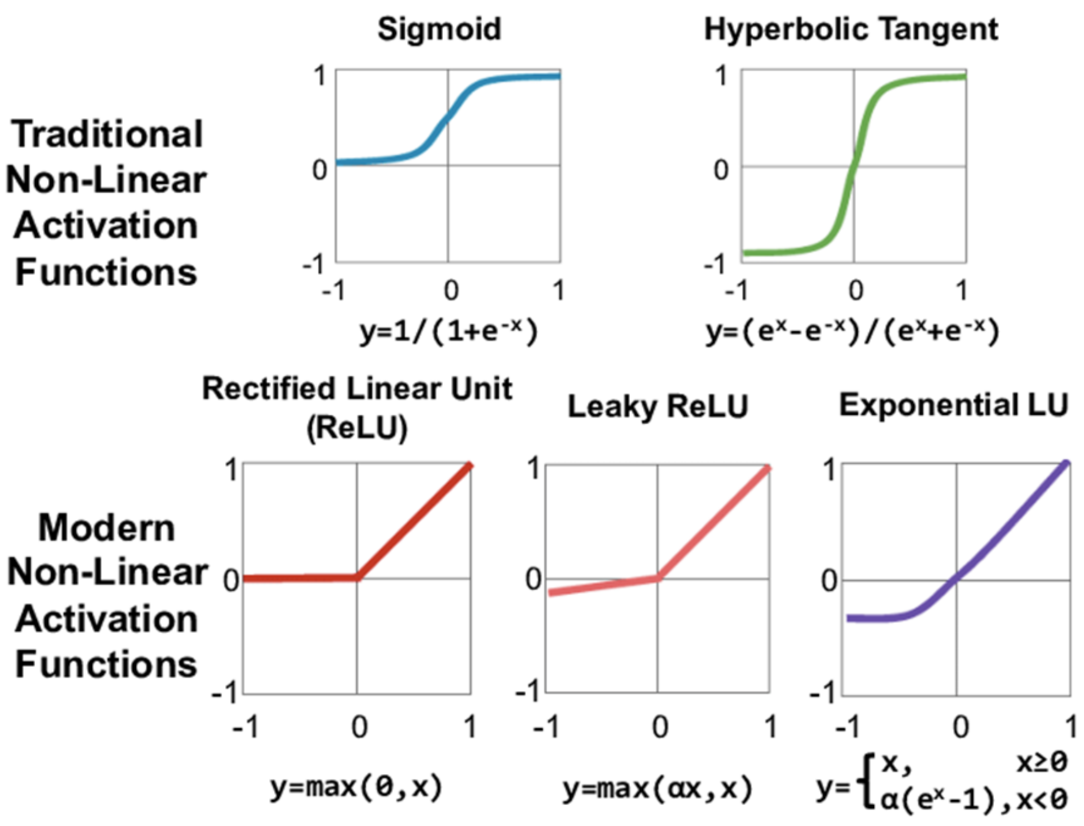

Figure 2.7: Activation functions

$$\overline{a^{(l)}} = \overline{W^{(l)}} \cdot \left(\overline{W^{(l-1)}} \cdot \overline{a^{(l-2)}} + \overline{b}^{(l-1)} \right) + \overline{b}^{(l)} = \overline{W'} \cdot \overline{a^{(l-2)}} + \overline{b'} \tag{2.23}$$

The use of this activation function for the hidden layer is useless. However, it is used in the output layer, especially in regression problems where the outputs are real numbers that can have any value. In this case, the output would be a linear combination of the weights and activations of the last hidden layer.

Sigmoid function

The sigmoid function, defined in Equation (2.24), is one of the most popular activation functions. It is represented in the top left of the Figure (2.7) and is characterized by providing values between 0 and 1. It is usually used as the activation function of the last layer in classification problems as the output can be considered as a probability. For example, if a model identifying if a person appears or not in a picture is considered, the desired output should 1 if there is a person or a 0 if not. This type of model uses the sigmoid function and the output is going to be considered as 1 if the output value is higher than 0.5 or 0 if it is lower.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.24}$$

Hyperbolic Tangent function

A most traditional choice for the activation function of the hidden layers is the hyperbolic tangent, defined in Equation (2.25). It is similar to the sigmoid function but it outputs values between -1 and +1, which allows some neurons to have zero value while having a mean value of zero. It is considered to be better than the sigmoid function for hidden layers as it allows the neurons to have negative values, improving the quality of the layer. On the contrary, it is never used in the output layer as its behavior does not add any good property to the model.

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.25}$$

The main disadvantage of this function, as for the sigmoid, is that its gradient is close to zero when the input values are far from zero. When training the network, the initial weights are randomly initialized and the values are far from the optimal. That implies that the gradients in these initial steps are going to be very small and the variation of the parameters of the network is going to be small too, making the training procedure longer. To speed up the training, some other activation functions are usually preferred, like the ReLU or its variations.

Rectified Linear Unit (ReLU)

The most used activation function nowadays is the Rectified Linear Unit (ReLU), defined as Equation (2.26) and showed in the bottom left side of Figure 2.7. It outputs the input value if it is bigger than 0 or 0 otherwise. The best quality of this function is that its gradient is always 1 for positive values and 0 for the negative ones, making faster the calculation of the gradients for the backpropagation algorithm. One may think that the discontinuity of the derivative in 0 can cause a problem to the network but it actually does not, as the value is never going to be exactly equal to zero due to the limited machine precision.

$$\sigma(x) = max(0, x) \tag{2.26}$$

As the gradient is always equal to 1 for positive values, the training procedure is going to be faster than the one using the tanh or the sigmoid functions. The fact of having 0 values for some of the neurons is not necessarily a problem, as it helps fighting overfitting, reducing the number of effective parameters of the model. However, the neurons getting 0 value are chosen in a random way, which is not a desirable thing. To avoid this problem, some variations of the ReLU function can be used.

Leaky ReLU and Exponential ReLU

These activation functions, showed in the bottom right side of Figure 2.7, are improvements of the original ReLU function trying to avoid the zero values assigned to some random neurons. The Leaky ReLU uses a linear function with a very small slope, whose value needs to be chosen manually, for the negative values. Furthermore, the Exponential LU uses a linear function for the positive values and an exponential function for the negative ones, working in a similar way than the original ReLU but assigning an almost constant value different than 0 for big negative values. Using this function raises the calculation time as the gradient is a function of the input value and needs to be calculated for each point.

These two activation functions present an extra hyperparameter α that needs to be tuned manually. The usual procedure to chose it is to try different values, observe how the network responds to them and chose the one that provides better performance. This task may take a lot of time and the benefits of using them are not very important for most of the cases, so the original ReLU is usually preferred.

2.4 Convolutional Neural Network (CNNs)

Convolution Neural Networks (CNNs or ConvNets) are a type of Artificial Neural Networks used typically in computer vision problems, due to the complexity of the task. If one tries to create an ANN to predict if there is a face in a picture or not, the data to be fed to the network are going to be 3 different 64x64 matrices, one for each of the RGB colors: Red, Green and Blue. To have all this information in the input layer of the network, these data need to be reshaped into a single vector of size 64x64x3 = 12288. That means that the network will have 12288 neurons on the first layer and a single neuron on the output layer, that will have a sigmoid activation function (section 2.3.2). If a first hidden layer of 1000 neurons is considered, the weight matrix of the first layer will have a dimension of 12288x1000, which corresponds to a total number of parameters of more than a 10 million only on the first layer. This enormous number of parameters is too big to be treated properly and a lot of data are necessary to train the network and avoid overfitting. This behavior is even more evident is higher resolution pictures are considered. For example, if a picture in 4K (4096x2160 pixels) is considered, the number of neurons of the first layer raises to more than 26 million.

Solving this problem is what Convolutional Neural Networks were developed for. They are based in the convolution operation, usually noted as $f * g$, that expresses how the shape of a function f is modified by other function g. The input of this network is a number of matrices, called channels, of the same size that can be, for example, 3 matrices of 64x64 elements representing each of the colors on each pixel. In the case of Black &

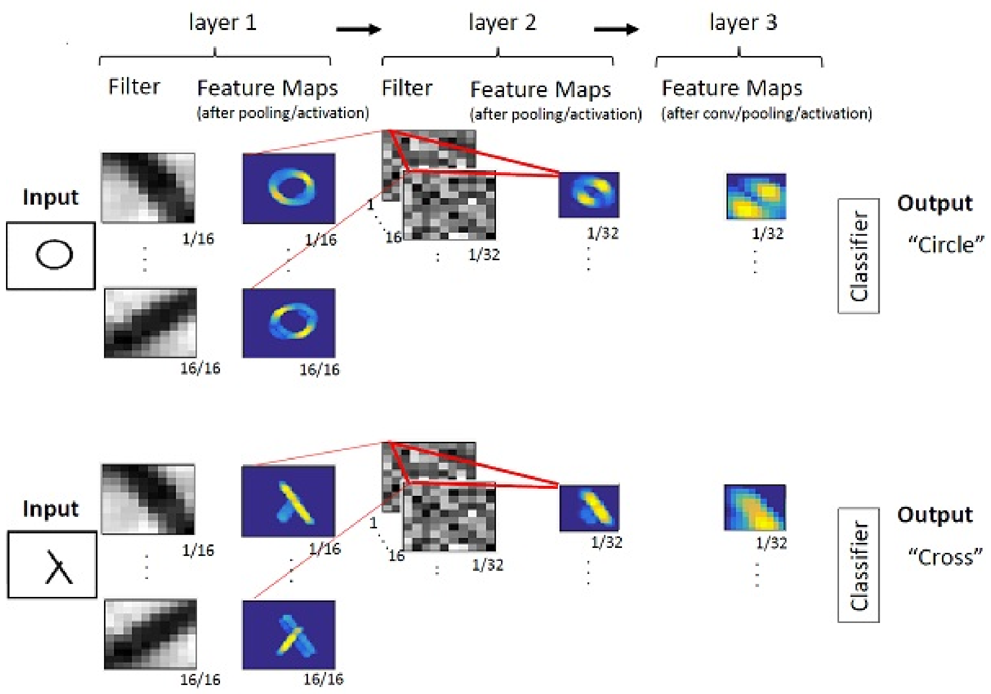
layer 1
layer 2
layer 3
Filter
Feature Maps (after pooling/activation)
Filter
Feature Maps (after pooling/activation)
Feature Maps (after conv/pooling/activation)
Input
1/16
16/16
1/32
Classifier
Output
"Circle"
"Cross"

Figure 2.8: Different layers of a CNN

White pictures, only one channel is considered.

Each one of these input channels is convoluted with a series of filters with a typical size of 3x3xc or 5x5xc, where c is the number of input channels, that moves around the whole picture and extract some local features. Figure 2.8 shows an example of a simple CNN that predicts whether the input image is a circle or a cross. Here, the filters represent the different possible orientations of the edges: horizontal, vertical, 40°, -20°, etc. The result of the convolution of each filter is called a feature map, that indicates the zone of the input picture that is more similar to the filter. In Figure 2.8, in the case where the input is a cross, it is clear how the feature maps work. When using the descending filter, the corresponding par of the cross gains more importance, whereas a similar map is obtained when using the ascending filter.

Now, there are c feature maps, all of them with the same dimension, which depends on the filter used and the strategy followed to move the filters around the input images. This is explained in section 2.4.1. New filters of size 3x3xn, where n is the number of filters used in the previous convolution, are now used over the feature maps previously obtained. They extract now information more general about the input images, such as edges, contours or intersections and new feature maps are again extracted. Following this procedure a few times, information about the global image can be extracted and the network is able to classify correctly the input image.

The filters of these deeper layers are less and less intuitive to humans in most cases. A typical example explaining this concept is getting to decide if a cat appears in a picture or not. The first layer would detect some local edges without any particular relevance. The second layer would look for some longer edges, connecting the previously found ones. The third layer would try to qualify these edges into the eyes, the ears, the mouth and so on. The following layer would try to put the different parts together, with the nose being between the eyes and the mouth, and with the ears being over the eyes. However, if this filters as observed, they make no sense at all to humans as they represent the importance of the previous features on each pixel, that have nothing to do with the original input image. With all these features, the input image is processed and all these filters are applied to it. If the filters detect the presence of some of the desired features, the image is classified as having a cat.

The key then is to have the correct filters that are able to extract as much important information for the considered problem as possible. Training a Convolutional Neural Network corresponds to finding the correct filters that are best suited for the task. This process is done automatically with the backpropagation algorithm, explained in section 2.3.1.

The key of the CNNs is that only the parameters of the filters need to be tuned. For example, if 10 filters of 5x5x3, for the 3 input channels of a color image, only 750 parameters need to be learned. For the next layer, 20 filters of 5x5x10, as there are 10 different feature maps, can be used. In this case, only 5000 parameters need to be learned. These numbers are very small compared to the millions of parameters required for each layer if a typical ANN would have been used.

2.4.1 Theory of CNN

The advantage of the CNN lies in the importance of the information extracted by all of the filters of the network. They perform a convolution over the input channels and the feature maps. A scheme of this operation is shown in Figure 2.9, where a filter of 3x3 pixels has been used.

The central pixel of this 3x3 filter is placed in the (2,2) position of the input layer and an element-wise multiplication is performed, obtaining a scalar that is placed in the position (2,2) of the feature map. Then, the same filter is switched to the position (2,3) of the input and the output is stored in the position (2,3). This operation is done for every position of the input layer, obtaining a feature map that is going to be smaller than the input layer. This is caused because the center of the filter can not be placed in the border of the input layer as there are not enough pixels to perform the convolution. With a filter of 3x3 pixels, the feature map is going to have 2 rows and 2 columns less than the input layer. For example, it will be a 62x62 matrix for a 64x64 input image. If a 5x5 pixels filter

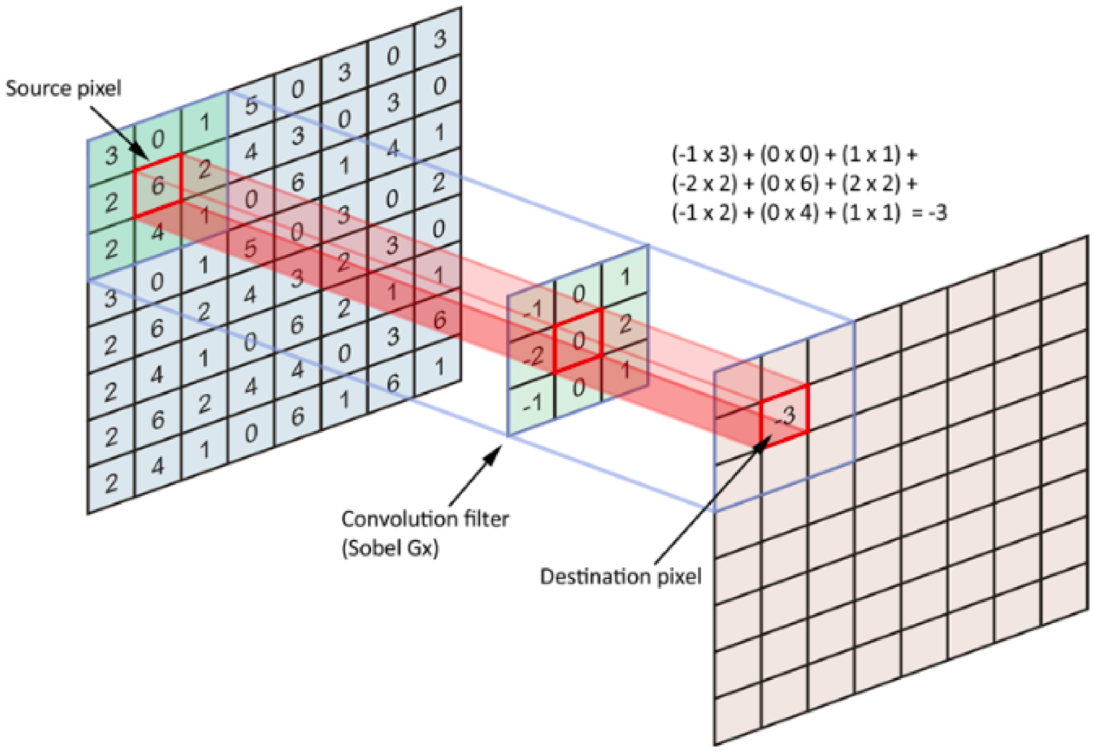
Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.9: Convolution operation

is used, the feature map would be 60x60.

This reduction of the size of the matrices is usually desired as it reduces the number of parameters of the network, which helps fighting overfitting, a typical problem of this type of algorithms. This also limits the number of layers that can be used. If this phenomenon wants to be avoided, a technique called padding can be used. Basically, it assigns values to the missing borders of the feature map in order to make it have the same size as the input. The assigned values can be zeros, ones, random values or the value of the closest pixel among other options.

At the end of the network, after all the convolutional layers, some extra layers are usually used to produce the desired output. For example, in the example of Figure 2.8, the desired output is an integer with a value of 1 referring to the circle or a value of 0 for the cross. At the end of the convolution, there are a series of several feature maps whose dimension is usually much smaller than the input image. All these pixels are flattened and are used as an input of a traditional fully connected neural network, like the one showed in Figure 2.6. Depending on the number and the size of the feature maps, some additional hidden layers can be added to get an output layer that provides the desired output. For the considered case, this last layer would have only one neuron and the activation function would be a sigmoid.

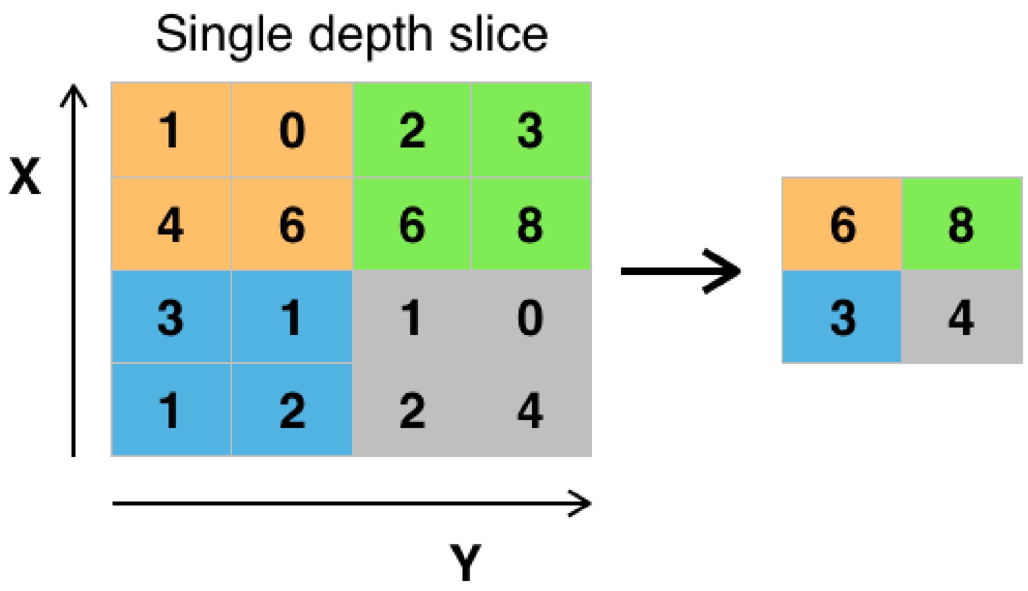

Figure 2.10: Maximum pooling layer with a 2x2 filter and a stride of 2

A conventional CNN as the ones explained so far has still a lot of data in each of the layers, having a lot of feature maps that need to be stored and operated. To reduce the dimensionality of this data, different techniques can be used. One of them is the stride that consists of skipping some positions while moving the filters during the convolution. Typically, the filter, instead of being moved one pixel in one direction, it is moved s pixels, where s is the stride of the layer. For 3x3 filters, a typical stride of 1 is used, whereas a stride of 2 is preferred for 5x5 filters.

Another technique is to add some pooling layers to the network. These layers perform a non-linear down-sample of the data. The most common pooling layer reduces the input data to a quarter of its size, half on each direction, by using 2x2 filters with a stride of 2. The activation function of this pooling layer can be anyone but the most used one is the maximum function. It examines a zone of 2x2 pixels and outputs the maximum value, as shown in Figure 2.10.

Chapter 3

Lattice Boltzmann Method

3.1 Mesoscopic scale

Traditional mathematical descriptions of fluid dynamics are based on the assumption that the time and length scales are sufficiently large that the atomistic picture can be averaged out. However, different approaches can be considered depending on the relevant length scale of the problem, as showed in Figure 3.1. From small to large, three characteristic length scales can be differentiated: the size of the fluid molecule l_a, the mean free path l_{mfp} (distance travelled for a molecule between two collisions with other molecules), scale of variation of macroscopic variables and its gradients l and the size of the considered problem l_s. Typically, they are ordered as $l_a << l_{mfp} << l \leq l_s$.

The microscopic simulations refer to a molecular description of the fluid, whereas the macroscopic ones considered the fluid as a continuum picture, with tangible quantities such as fluid velocity or density. The microscopic systems are ruled by Newton's dynamics and the macroscopic ones by the Navier-Stokes equations. In between, there exits the mesoscopic description that tracks distributions or representative collections of molecules. It uses the Kinetic Theory (section 3.2) to express mathematically the phenomena involved.

A similar hierarchy, coupled to the length, can be made for the time scales. At very short times one can define the collision time t_c: the duration of a collision event. In the standard kinetic energy, the collisions are supposed to be instantaneous and $t_c \to 0$. Next, the mean flight time can be defined as the time between two successive collisions: $t_{mfp} = l_{mfp}/v_T$, where $v_T = \left(\frac{k_B T}{m}\right)^{1/2}$ is the thermal velocity of the molecule, k_B being

This chapter has been based in the book from Krüger [12] and in the lecture presentations of professor N. Gourdain at ISAE-Supaero

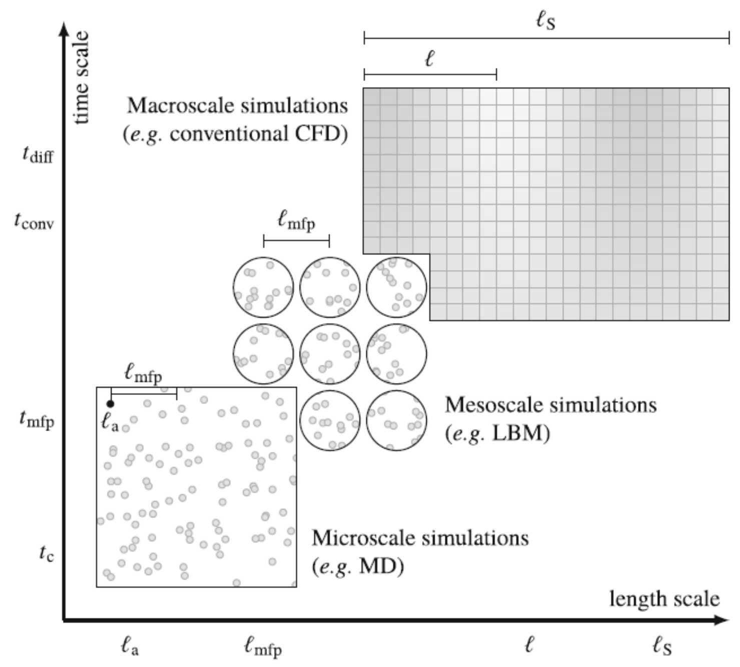

Figure 3.1: The hierarchy of length and time scales in typical fluid dynamics problems

the Boltzmann's constant and m the mass of the gas molecule, in a way that $R = k_B/m$, R being specific gas constant. This thermal velocity is of the order of the speed of sound and much bigger than the macroscopic velocity of the flow and represents the velocity of a molecule in the fluid due to the thermal agitation induced by being at a temperature higher than 0K. It is on this scale where the kinetic theory operates and where the system relaxes to a local equilibrium through collisions of molecules. This local equilibrium does not mean that the system is in global equilibrium, and the opposite case is actually most likely to occur.

With longer times and larger length scales, there can be a flow from one part of the fluid to another. Two mechanisms can cause this phenomena: convection(inertial regime) or diffusion (viscous regime). The importance of each term and, therefore, the shortest time scale, is measured by the Reynolds number, defined as:

$$Re = \frac{t_{diff}}{t_{conv}} = \frac{l^2/\nu}{l/u} = \frac{ul}{\nu} \tag{3.1}$$

where ν is the kinematic viscosity. Flows with high Reynolds number are the most usual cases as they correspond tho flows around planes or vehicles but there is also a growing interest is low Reynolds flows are they are used in micro-physics and biophysics.

A larger scale can also be defined as the acoustic time scale: $t_{sound} = l/c_s$, where c_s is the speed of sound in the fluid. This scale represents how fast the compression waves travel through the future. If it is faster than the advective time scale, the fluid can be considered as incompressible; otherwise, there appears compressible phenomena that can induce the formation of more complex phenomena, such as shock waves. This ratio is called the Mach number, defined as $M = t_{sound}/t_{conv} = u/c_s$. In practice, a limit of $M \leq 0.3$ is usually chosen as the impressibility limit.

Another useful parameter is the Knudsen number, defined as the ratio between the mean free path length and the characteristic size of the problem l: $Kn = l_{mfp}/l$. If the Knudsen number is very small, the Navier-Stokes equations are valid, whereas its validity stops for $Kn \geq 1$, where only the general kinetic theory remains valid.

3.2 Kinetic Theory: Boltzmann equation

The kinetic theory is a fluid description that lies in the mesoscopic scale and that describes the distribution of particles in a gas, quantity whose characteristic time scale is of the order of the free mean path time t_{mfp}. This theory is usually used for dilutes gases where the molecules spend very little time colliding to each other: $t_c \ll t_{mfp}$. This assumption implies that almost never three particles are going to be simultaneously involved in the same collision, supposing all the collision one-to-one. In addition, only mono-atomic gases are considered as single atoms collide elastically, thus conserving all the translational energy. This is made for simplicity as molecules with more than one atom have additional inner degrees of freedom, such as vibrations and rotations, that make the collision non-elastic. Nevertheless, the macroscopic behavior of mono-atomic and poly-atomic gases is largely similar.

The distribution function and its moments

The most important variable in the Kinetic Theory is the particle distribution function $f(\vec{x}, \vec{\xi}, t)$, where $\vec{\xi}$ represents the microscopic particle velocity. This function represents the density of particles with velocity $\vec{\xi} = (\xi_x, \xi_y, \xi_z)$ at position $\vec{x}$ and time t. It represents the density of mass in the three dimensions of the velocity space, thus having dimensions of $[kg \cdot s^3 \cdot m^{-6}]$.

The distribution function represents the macroscopic variables through its moments: integral of f, weighted with some function of $\vec{\xi}$, over the entire velocity space. For instance, the mass density can be calculated as:

$$\rho(\vec{x}, t) = \int f(\vec{x}, \vec{\xi}, t) \; d^3\xi \tag{3.2}$$

where the contributions of particles at position $\vec{x}$ and time t with all the possible velocities have been added up. In addition, the sum of the particles' contributions $\vec{\xi} \cdot f$ to the momentum density can also be calculated as:

$$\rho(\vec{x},t)\vec{u}(\vec{x},t) = \int \vec{\xi} \cdot f(\vec{x},\vec{\xi},t)\ d^3\xi \tag{3.3}$$

Similarly, the macroscopic total energy density can be obtained from:

$$\rho(\vec{x},t)E(\vec{x},t) = \frac{1}{2}\int \|\vec{\xi}\|^2 \cdot f(\vec{x},\vec{\xi},t)\ d^3\xi \tag{3.4}$$

The total energy is composed of two different types of energy: the one due to the bulk motion of the fluid, $\frac{1}{2}\rho\|\vec{u}\|^2$, and the internal energy due to the random thermal motion of the gas particles. A relative velocity v can be defined as the difference between the local motion of each particle and the bulk velocity: $\vec{v}(\vec{x},t) = \vec{\xi}(\vec{x},t) - \vec{u}(\vec{x},t)$. This allow to express the internal density energy as:

$$\rho(\vec{x},t)e(\vec{x},t) = \frac{1}{2}\int \|\vec{v}\|^2 \cdot f(\vec{x},\vec{\xi},t)\ d^3\xi \tag{3.5}$$

For monoatomic ideal gases moving in a three-dimensional space, it can be probed that:

$$p = \rho RT = \frac{2}{3}\rho e = \frac{1}{3}\int \|\vec{v}\|^2 \cdot f(\vec{x},\vec{\xi},t)\ d^3\xi \tag{3.6}$$

All these equations relate the macroscopic variables, such as pressure, velocity, temperature, mass density, etc. with the mesoscopic ones. This means that the solution of the mesoscopic scales provides the solution of the macroscopic ones, which are the ones that interest the most for engineering purposes.

The Equilibrium Distribution Function

When two solid spheres collide, the outgoing velocity of each one of them is very sensitive to small changes in their relative positions. However, collisions tend to even out the angular distribution of particle velocities in a gas around the mean velocity $\vec{u}$. This means that if a gas is left alone for sufficiently amount of time, its distribution function $f(\vec{x},\vec{\xi},t)$ will eventually reach an equilibrium distribution $f^{eq}(\vec{x},\vec{\xi},t)$, that is going to be isotropic in velocity space around its mean velocity $\vec{\xi} = \vec{u}$. In a reference frame moving at velocity $\vec{u}$, this equilibrium distribution function can be expressed as:

$$\boxed{f^{eq}(\vec{x},\|\vec{v}\|,t) = \rho \cdot \left(\frac{1}{2\pi RT}\right)^{2/3} \cdot e^{-\|\vec{v}\|^2/(2RT)}} \tag{3.7}$$

In addition, the equilibrium distribution function can be developed with Taylor series, where only terms up to order 4 are necessary to achieve the Navier-Stokes equations (mass, momentum and energy). The equilibrium distribution function gets then reduced to:

$$\begin{aligned} f^{eq}(\vec{x}, \|\vec{v}\|, t) = \rho \cdot \left(\frac{1}{2\pi RT}\right)^{2/3} \cdot exp\left(-\xi^2/(2RT)\right) \cdot \\ \cdot\, exp\left[1 + \frac{(\xi u)}{RT} + \frac{(\xi u)^2}{2(RT)^2} - \frac{u^2}{2(RT)} + \frac{(\xi u)^3}{2(RT)^3} - \frac{(\xi u)\cdot u^2}{2(RT)^2} + \mathcal{O}(u)^4\right] \end{aligned} \tag{3.8}$$

The Boltzmann Equation

In order to solve the problem, the evolution of the distribution function needs to be calculated, and, to do so, an equation indicating how it evolves in time is required. This equation is known as the Boltzmann equation and is expressed as:

$$\frac{\partial f}{\partial t} + \xi_\beta \frac{\partial f}{\partial x_\beta} + \frac{F_\beta}{\rho} \cdot \frac{\partial f}{\partial \xi_\beta} = \Omega(f) \tag{3.9}$$

The first two terms of the equation represent how the distribution function is advected with the velocity $\vec{\xi}$ of its particles. The third term represents the forces $\vec{F}$ affecting this velocity. $\Omega(f)$ is called the collision operator and can be interpreted as a source term that represents the local distribution of f due to the collisions. As the collisions conserve the mass, the momentum and the translational energy (in monoatomic gases), the moments of the collision operator must be zero.

The collision operator requires a modeling to represent all the possible collisions between particles that can exist, resulting in very complex models like the one proposed originally by Boltzmann. However, the LBM is usually based on the much simpler BGK collision operator model, name after its inventor Bhatnagar, Gross and Krook. It can be expressed as:

$$\Omega(f) = -\frac{1}{\tau} \cdot (f - f^{eq}) \tag{3.10}$$

The BGK model expresses the relaxation of the distribution function towards the equilibrium distribution. The speed at which this state is reached is expressed by the relaxation time τ, which determines the transport coefficients like the viscosity or heat diffusivity. This model is the simplest one can get and it is not as exact as others, like the one proposed by Boltzmann, thus providing less accurate solutions. For example, the BGK model predicts a value of the Prandtl number of $Pr = 1$, whereas the Boltzmann model correctly predicts $Pr \simeq 2/3$. If a spatially homogeneous case, where the problem is independent of the position, and without any external force applied, is considered, the distribution function evolves towards the equilibrium exponentially. One can show that this tendency can be expressed as:

$$f(\vec{\xi},t) = f^{eq}(\vec{\xi}) + \Big(f(\vec{\xi},0) - f^{eq}(\vec{\xi})\Big) \cdot e^{-t/\tau} \tag{3.11}$$

The Boltzmann Equation (3.9) can describe the macroscopic behavior of a fluid if their moments are considered. If an approximation of $f \approx f^{eq}$ is made, one can find the equations conforming the Euler model. If a first-order approximation $f \approx f^{eq} + \epsilon f^{(1)}$ is considered, one can find the Navier-Stokes equations. Higher-order approximations are usually not considered as they do not improve the accuracy of the solutions.

One can also relate the entropy to the distribution function f with the so-called Boltzmann's $\mathcal{H}$-Theorem. It can be showed that the quantity $\mathcal{H}$, expressed in Equation (3.12), can only ever decrease and reaches its minimum value at the equilibrium state. This behavior is analogous to the entropy of a system, that always increases until it reaches its maximum value, corresponding to the equilibrium state. In fact, it can be probed that $\mathcal{H}$ is proportional to the entropy density: $\rho \cdot s = -R \cdot \mathcal{H}$

$$\mathcal{H} = \int f \cdot ln(f)\; d^3\xi \tag{3.12}$$

In summary, supposing that no external forces act in the problem, the Boltzmann equation with the BGK model for the collision operator can be expressed as Equation (3.13), where f^{eq} is expressed in Equation (3.7). This is a simple hyperbolic equation that represents the advection of f with the particle velocity $\vec{\xi}$. The assets of this formulation rely on the linearity of this advection term, with all the non-linearities contained in the collision term. In addition, this collision term depends only on the local value of f and not on its gradients, which allows the use of much simpler numerical schemes that are easy to implement and to parallelize.

$$\boxed{\frac{\partial f}{\partial t} + \vec{\xi}\nabla f = -\frac{1}{\tau}\left(f - f^{eq}\right)} \tag{3.13}$$

3.3 The Lattice Boltzmann Equation

The previous equations express the evolution of the distribution function. However, this variable is continuous, as particles can move in all directions, and analytical solutions are almost impossible to achieve. In practice, this equation must be discretized in velocity, time and space, prescribing a set of possible velocities for the particles to move with. In addition, it has to be noticed that the full knowledge of the distribution function is not mandatory in order to recover the macroscopic variables, as they only depend on its moments.

The distribution function can be expressed as a sum of Hermite polynomials H_n, where the coefficients a^n are the moments of f, which correspond to the macroscopic magnitudes.

$$f = \frac{1}{(2\pi)^{D/2}} \cdot e^{-\xi^2/2} \cdot \left[\prod_{i=1}^{D} \left(\sum_{n=0}^{\infty} \frac{a_i^{(n)}}{n!} \cdot H_n(\xi_i) \right) \right] \tag{3.14}$$

$$\begin{aligned} a^{(0)} &= \int f \, d\xi & a^{(1)} &= \int f \cdot \xi \, d\xi \\ a^{(2)} &= \int f\xi\xi \, d\xi & a^{(3)} &= \int f \cdot \xi\xi\xi \, d\xi \end{aligned} \tag{3.15}$$

Each of the coefficients can be calculated thanks to the orthogonality of Hermite polynomials, evaluating the integral through a quadrature in the velocity space:

$$a^{(n)} = \int f(\vec{x}, t, \vec{\xi}) \cdot H_n(\vec{\xi}) \, d\vec{\xi} \approx \sum_{k=1}^{d} w_k \cdot (2\pi)^{D/2} \cdot e^{\xi^2/2} \cdot f^N(\vec{x}, t, \xi_k) \cdot H_n(\xi_k) \tag{3.16}$$

The order of the quadrature implies a discrete set of velocities ξ_k, each of them with a weight w_k. f^N refers to the truncated Taylor series of f up to the order N ($N = 4$ to obtain the Navier-Stokes equations). Now, instead of having a continuous equation for the distribution function f, there is a set of discrete ones f_k, representing the population of particles moving with velocity ξ_k, in a way that the moments of order N are conserved:

$$\boxed{\frac{\partial f_k}{\partial t} + \xi_k \cdot \nabla f_k = \frac{1}{\tau} \cdot (f_k^{eq} - f_k)} \tag{3.17}$$

To recover the Navier-Stokes equations, with f up to order 4, a Gauss-Hermite quadrature with 25 (2D) or 125 (3D) points are needed. However, to overcome the cost of solving 125 independent equations for the 3D case, the energy equation can be omitted, which is equivalent to an order 3 for the development of f, reducing it to a set of only 27 velocities. This induces an error on the equilibrium distribution function, that corresponds to an error of order $\mathcal{O}(M^3)$ in the momentum equation, thus limiting the validity of the model to athermal and weakly compressible flows (typically $M < 0.3$). For flows with higher Mach numbers, a set of more velocities has to be used to capture all the physic phenomena.

Following a similar reasoning, different discretizations of the velocity can be considered. They are usually named 'DdQq', where 'd' corresponds to the number of spatial dimensions considered and 'q' to the number of velocities. The most used one in three-dimensional problems is the D3Q19, showed in Figure 3.2, that provides a good compromise between computational time and numerical stability. In addition, it has to be noticed that each individual velocity has an assigned weight, that depends on the distance from the center of a cube to its surface on the considered direction. A table with several different discretizations, with its corresponding directions and weights, is showed in Table 3.1.

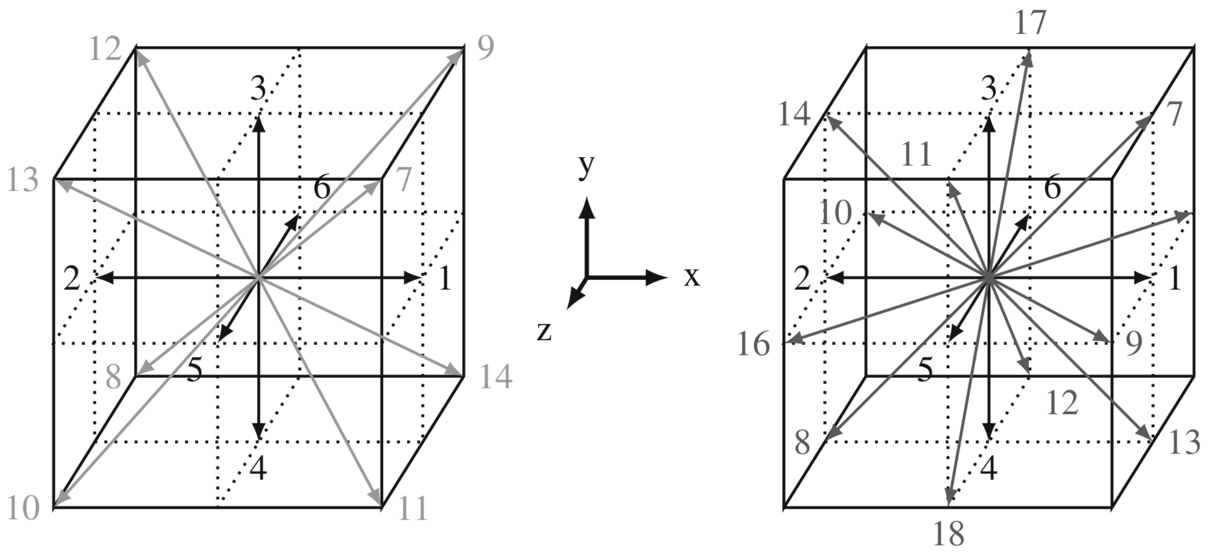

Figure 3.2: D3Q19 Two different velocity discretization: D3Q15 on the lift side and D3Q19 on the right side

Table 3.1: Most used velocity discretizations

Notation	Velocities $\boldsymbol{c}_i$	Number	Length $\lvert\boldsymbol{c}_i\rvert$	Weight w_i
D1Q3	(0)	1	0	2/3
	(±1)	2	1	1/6
D2Q9	$(0,0)$	1	0	4/9
	$(\pm1,0)$, $(0,\pm1)$	4	1	1/9
	$(\pm1,\pm1)$	4	$\sqrt{2}$	1/36
D3Q15	$(0,0,0)$	1	0	2/9
	$(\pm1,0,0)$, $(0,\pm1,0)$, $(0,0,\pm1)$	6	1	1/9
	$(\pm1,\pm1,\pm1)$	8	$\sqrt{3}$	1/72
D3Q19	$(0,0,0)$	1	0	1/3
	$(\pm1,0,0)$, $(0,\pm1,0)$, $(0,0,\pm1)$	6	1	1/18
	$(\pm1,\pm1,0)$, $(\pm1,0,\pm1)$, $(0,\pm1,\pm1)$	12	$\sqrt{2}$	1/36
D3Q27	$(0,0,0)$	1	0	8/27
	$(\pm1,0,0)$, $(0,\pm1,0)$, $(0,0,\pm1)$	6	1	2/27
	$(\pm1,\pm1,0)$, $(\pm1,0,\pm1)$, $(0,\pm1,\pm1)$	12	$\sqrt{2}$	1/54
	$(\pm1,\pm1,\pm1)$	8	$\sqrt{3}$	1/216

Once the velocity discretization has been selected, the space and time also need to be discretized. One of the limitations of the LBM is that the cells must have a cubic shape in order to allow the previous velocity discretization. So, only the choice of Δx is required. In addition, the distribution function is moving to a distance Δx in with a velocity ξ_i, so the required time is $\Delta t = \Delta x / \xi_i$, which determines the time discretization.

Usually, the previous equations are normalized before being solved in a way that $\Delta x = \Delta t = 1$, which implies a normalization of the pseudo speed of sound to $\widetilde{\xi_0} = \sqrt{RT} = 1/\sqrt{3}$. It also fixes the CFL number to $CFL = \xi_0 \cdot \Delta t / \Delta x = 1/\sqrt{3}$.

Once that each one of the discrete distribution functions have been calculated, one can easily return to the macroscopic variables with:

$$\begin{aligned}
\rho &= \sum_{k=0}^{q-1} f_k \\
\rho u &= \sum_{k=0}^{q-1} \xi_k f_f \\
\rho e + \frac{1}{2}\rho u^2 &= \frac{1}{2}\sum_{k=0}^{q-1} \xi_k^2 f_k
\end{aligned} \tag{3.18}$$

3.4 Numerical scheme to solve the LBGK equation

As explained before, from the Boltzmann equation and using the BGK model for the velocity discretization, one finds a set of Q equations, each one corresponding to each single discretized velocity. If $i \in [0, Q-1]$ denotes each one of the velocities, the remaining Lattice-Boltzmann-BGK (LBGK) equations can be expressed as:

$$\boxed{f_i(\vec{x} + \vec{e_i}, t + \Delta t) = f_i(\vec{x}, t) - \frac{1}{\tau} \cdot \left[f_i(\vec{x}, t) - f_i^{(eq)}(\vec{x}, t) \right]} \tag{3.19}$$

The equation can be separated into two parts: one corresponding to the collisions between particles at a given time instant, and the other to the advection of these particles after the collisions. To solve the equations the following procedure, depicted in Figure 3.3, has to be followed:

1. The first step is the calculation of the equilibrium distribution function $f_i^{(eq)}$ from the macroscopic density and velocity fields using the Equation (3.8). To initialize the variables, values of 1 and 0 are usually given to the initial density and velocity fields, respectively.

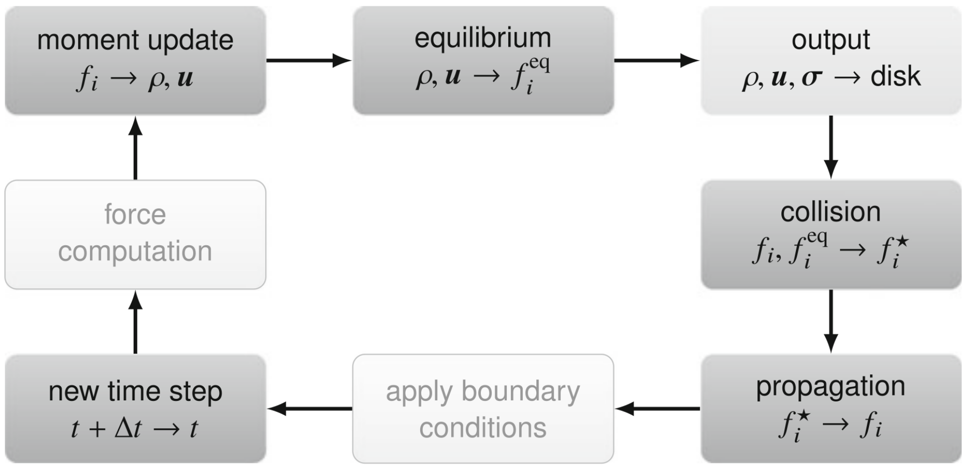
moment update
$f_i \rightarrow \rho, \boldsymbol{u}$
equilibrium
$\rho, \boldsymbol{u} \rightarrow f_i^{\mathrm{eq}}$
output
$\rho, \boldsymbol{u}, \boldsymbol{\sigma} \rightarrow$ disk
force computation
collision
$f_i, f_i^{\mathrm{eq}} \rightarrow f_i^{\star}$
new time step
$t + \Delta t \rightarrow t$
apply boundary conditions
propagation
$f_i^{\star} \rightarrow f_i$

Figure 3.3: Schema of one cycle of the Lattice Boltzmann algorithm. The lighter boxes indicate optional steps

2. Perform the collisions (relaxation) between the different particles, calculating $f_i^{\star}$ as:

$$f_i^{\star}(\vec{x},t) = f_i(\vec{x},t) \cdot \left(1 - \frac{\Delta t}{\tau}\right) + f_i^{eq}(\vec{x},t) \cdot \frac{\Delta t}{\tau} \tag{3.20}$$

3. Perform the streaming (propagation) of the particles from one pint of the lattice to its neighbor in each direction. This is made with the equation:

$$f_i(\vec{x} + \vec{c_i} \cdot \Delta t, t + \Delta t) = f_i^{\star}(\vec{x},t) \tag{3.21}$$

4. Increase the time step, setting t to $t + \Delta t$.

5. Calculate the macroscopic magnitudes with the Equations (3.18).

6. Repeat the cycle until the convergence is reached.

3.5 Overview of the LBM

The Lattice Boltzmann Method previously explained may look like a very complicated method that does not provide any advantage with respect to other conventional methods of solving the Navier-Stokes equations. However, it possesses a number of properties that make this method one of the most active research fields in the CFD domain as it allows to solve very complex flows with an accuracy unreachable by the RANS based solvers, especially those with a low Mach number.

The main problem of the traditional CFD solvers is the calculation of the advection term $\vec{u} \cdot \nabla \vec{u}$, which is non-linear and involves an iterative process that requires the determination of approximate derivatives from the values of the velocity in adjacent nodes. In contrast, the particle-based formulation of the LBM produces a set of equations where the "non-linearity is local and the non-locality is linear". This means that all the non-linear terms fall into the collision term, which is local for each one of the nodes, whereas the advection term, that streams all the particles from one node to another, is local. These characteristics allow the equations to be easily parallelize into modern GPUs, that speed up the calculation time for the LBM. On the other hand, the propagation of the particles is very memory-intensive, needing to access the memory an enormous number of times, which is the main bottleneck of Lattice Boltzmann computations. In addition, the LBM is inherently time-dependant, reason why is not efficient for steady flow simulations.

The standard LBM can be seen as a second-order accurate solver for the weakly compressible Navier-Stokes equations, where weak compressibility refers to errors that become more and more relevant as the Mach number raises. The standard LBM formulation is then valid up to Mach numbers close to 0.3, where the compressibility terms start becoming important. However, the solutions provided by the LBM are more accurate and present less dissipation than the ones provided by conventional CFD methods, even the high-order ones, while keeping the dispersion errors similar. This low dissipation makes the LBM very suited for aeroacoustic problems and complex aerodynamics situations where the Mach number is low.

One of the main disadvantages of the LBM is that it needs a cubic discretization, with all the cells being squares of the same dimensions. This may cause problems when dealing with complex geometries as it can not adapt very well to the walls. However, is very well suited for mass-conserving flows in complex situations, like porous media. In addition, it is well suited to solve multi-phase flows, although there are some unsolved problems that prevent the method to provide any improvement with respect to conventional CFD solvers.

Chapter 4

Methodology and Results

4.1 Overview of the Methodology

The main goal of this project is the creation of a model, based on Convolutional Neural Networks, to calculate pressure fields in the transonic regime. The idea is to extend the Prandtl-Glauert similarity rule to transonic regimes with less restrictive hypothesis: the model would take a low-Mach number pressure field and would apply the learnt transformations to predict the corresponding field at high-Mach number, with the shock waves and detachments of the boundary layer that might appear.

The methodology followed in this work can be divided into three different parts. Firstly, a database containing a large number of relevant cases for the problem needs to be created. To do so, one could use experimental data, but its availability and the amount of money and time that they require make this task almost impossible. Numerical simulations have been used instead as they are easier to perform and allow to have a wide range of values for all the variables. This is explained in section 4.2.

A Convolutional Neural Network is trained with this database of numerical cases to learn the physical relationships involved in the transformation from low to high-Mach number flows. Once trained, the model should be able to accurately predict pressure fields at high velocities from incompressible fields. The creation of this model is explained in section 4.3.

LBM simulations can be used as an alternative to the RANS simulations as inputs of the Neural Network. For instance, [6] probes that this method provides more accurate solutions than traditional RANS-based solvers. However, its standard formulation is only valid in the weakly compressible regime, up to $M < 0.3$, due to the few velocities used in the discretization [21] in order to get solutions in a reasonable amount of time. The idea is, then, to calculate highly accurate incompressible solutions with the LBM and input them into the network: it will apply the learned transformations and calculate the pressure field

FROM	TO			
0.1	0.2	0.3	0.4	0.5
0.2		0.3	0.4	0.5
0.3			0.4	0.5
0.4				0.5

Table 4.1: Example of creation of the couples to be fed to the network

for any higher Mach number, as discussed in section 4.4.

4.2 Database creation

The crucial step when developing ML-based surrogate modelling is to collect a large and accurate database. To do so, numerical simulations have been performed as they provide accurate solutions in a relatively short amount of time. It will require a preliminary validation on the numerical simulations against known experimental measurements to ensure a clean and accurate database generation.

The problem considered in this work is the bi-dimensional flow around a transonic airfoil with different angles of attack and velocities while keeping the Reynolds number constant with a value of $6.5x10^6$. The present study focuses on the RAE2822 geometry, a typical transonic airfoil whose numerical and experimental data are available.

The model to be created is inspired by the Prandtl-Glauert transformation (Equation (1.24)): for a given angle of attack, an incompressible field at low Mach number, easy to calculate, is used to calculate the corresponding compressible case at higher Mach. The data to be fed to the network are, then, couples of pressure fields, one at low Mach and its corresponding at high Mach with the same angle of attack. An example of the creation of these couples is shown in Table 4.1.

In order to create these couples, several simulations varying the Mach number in the subsonic regime, while keeping the angle of attack and the Reynolds number constant, are required. In addition, this same procedure needs to be repeated for different values of the angle of attack so as the network is able to understand its influence in the problem. The selection of the Mach and alpha distribution is very important as they have to cover the whole considered domain while ensuring that the model is independent of this choice.

Deep learning techniques usually require very large datasets to actually learn to solve the problem. However, the exact amount of data required is not known and is very dependant on the case of study. The way of selecting these data is probably of key interest to ensure a good quality of the Neural Network while limiting the number of data samples to be generated.

Following [9], two different distributions have been created. Firstly, a random distribution has been used, choosing 21 random values of the incidence and 50 random values of Mach, different for each case. This distribution needs a lot of points to effectively cover all the domain but is very robust and efficient for a very large number of parametres. Then, the Clenshaw-Curtis distribution, defined in Equation (4.1), has been considered, where $j_k \exists [1, n_k]$ corresponds to each considered point and $n_k = 2^p + 1$ is the total number of points. For this work, 33 values of the angle of attack and the Mach have been considered. This distribution ensures that all the domain is tested with relatively few points, whose number can be selected.

$$y_k^{j_k} = -cos(\frac{\pi \cdot (j_k - 1)}{n_k - 1}) \tag{4.1}$$

In order to perform all these different simulations, a variation of the code VLab2, created by prof. V. Chapin at the ISAE-Supaero, has been used. It writes different journal files with all the required information that are later read by the different external programs, the mesher Gambit 2.4 and Fluent 6.3 in this case. This allows to run several simulations on a loop, without the need of launching manually each individual one. The main goal of VLab2 is the shape optimization of aerodynamic objects in order to get their best aerodynamic behavior. To do so, the geometry can be automatically meshed with some desired parametres, like the minimum size of the cells or its growing ratio among others. In addition, a lot of several turbulence models are available, like the Spalart-Allmaras, the $k - \omega$ or the $k - \epsilon$. This code has already been used in several projects providing outstanding results like in [34].

Numerical model

Using a RANS approach is an affordable strategy to compute high Reynolds number cases at low cost. However, it requires the modeling of the turbulence, as explained in section 1.2. For this case, the Spalart-Allmaras model with a turbulent intensity of 0.1 and a turbulent scale of 0.005 has been used, as it provides accurate solutions for bi-dimensional flows around airfoils in a relatively short amount of time. A stationary case has been considered, as the number of cases with unsteady regimes is very small and they are more expensive to compute [14]. A second-order upwind discretization and a density-based implicit solver have been used, as they provide better solutions for compressible flows [37]. The air has been considered as an ideal gas with constant properties. The stopping criterion has been set to 5000 iterations as it has been proved that they are enough for the solution to converge. In addition, if the lift and drag coefficient variations between iterations is less than the 0.1%, the simulation is said converged and it is stopped to reduce the calculation time.

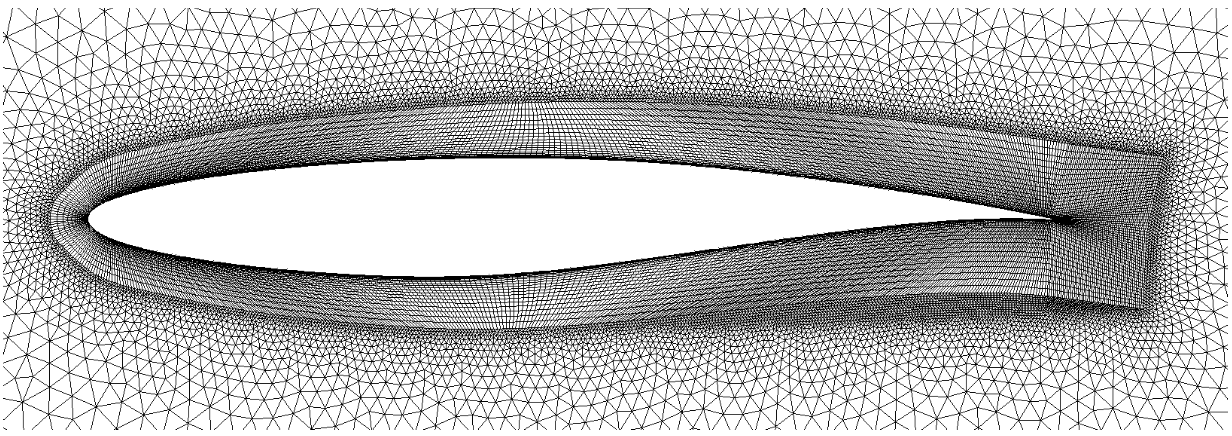

Figure 4.1: Mesh used for the creation of the database

Mesh

Even though that different discretizations could have been used for each individual simulations, an unique robust mesh has been utilized for all the cases in order to reduce the time of creating a different one for each case. An hybrid mesh has been used, with an structured part in the surroundings of the airfoil and with an unstructured part elsewhere. The total domain of calculation extends 20 chords on each direction from the leading edge of the airfoil, making a total surface of 40x40 chords.

The height of the structured part has been set to $1 \cdot 10^{-4}$ times the chord in the leading edge and it raises up to double its size in the trailing edge. The elements along the surface of the airfoil have been uniformly distributed with a size of $1 \cdot 10^{-3}$ times the chord. In the leading edge, the size of the elements has been reduced to its half to improve the resolution. In the direction normal to the surface, 25 elements have been considered with the first element having a height of $1 \cdot 10^{-4}$ and setting a growth rate of 1.15. A picture of the mesh is shown in Figure 4.1.

Some of the simulations carried out with these conditions diverge due to the irregularity of the mesh. To solve this issue, these cases have been re-simulated with a slightly refined mesh, raising the number of cells in the structured zone from 25 to 28 and reducing the height of the structured zone to $8 \cdot 10^{-5}$. This doubles up the calculation time, reason why this more refined mesh is only used for the cases that have diverged with the standard one.

Validation

To measure the accuracy of the solutions provided by the model and the mesh indicated previously, the results have been compared with the experimental measurements provided by [19]. The relative errors for each case are shown in Table 4.2. The error on the lift and momentum coefficients are less than the 8% whereas the drag coefficient raises up to almost 30%. This error is produced mainly for the chosen model, as it does not reproduce accurately the boundary layer, thus not providing the correct values for the friction in the airfoil's surface. However, the pressure coefficient, showed in Figure 4.2, is calculated very accurately, the main difference being the position of the shock wave when formed.

Mach	0.676	0.6	0.725	0.725	0.728	0.73	0.75	0.74	
α	2.40	2.57	2.92	2.55	3.22	3.19	3.19	3.19	
Re(E6)	5.7	6.3	6.5	6.5	6.5	6.5	6.2	2.7	Mean
Cl exp	0.566	0.522	0.743	0.658	0.802	0.803	0.743	0.733	Error
Rel error (%)	10.28	13.2	6.7	10.87	4.8	4.2	5.31	6.0	7.67
Cd exp	0.0085	0.0101	0.0127	0.0107	0.0175	0.0168	0.0242	0.0188	
Rel error (%)	22.69	6.54	34.9	24.4	27.3	32.8	21.75	43.4	28.9
Cm exp	0.082	0.073	0.095	0.09	0.1	0.099	0.0106	0.086	
Rel error (%)	2.03	3.64	7.3	2.3	8.9	5.85	6	3.41	4.93

Table 4.2: Comparison between experimental and numerical simulations for the RAE2822

In addition, the results have been compared with other numerical solutions provided by [38]. There, the authors calculate the lift and drag coefficients for a range of subsonic Mach numbers while keeping the incidence and the Reynolds number constant. The results are showed in Figure 4.3, where the solid line corresponds to Zhu's numerical solutions, the dashed line corresponds to the solution obtained applying the Prandtl-Glauert similarity rule (Equation (1.24)) and the red points are the ones calculated in this work. It can be seen that the results are very close to Zhu's thus validating the model and mesh used in this work. As expected, the Prandtl-Glauert transformation becomes exponentially inaccurate for Mach number above 0.7, which is a strong limitation of the method since typical transport aircraft fly at a cruise Mach number close to 0.8. This calls for an improvement of the existing method to predict high Mac number flows from incompressible simulations, which is the topic covered by the present study.

A study about the friction coefficient on the surface of the airfoil has also been performed as it is responsible for the viscous contribution of the drag. There are no experimental data measuring this coefficient as it is very hard to place sensors in the boundary layer. However, to validate the simulations, the friction coefficient calculated has been compared with the results obtained by R. Olivanti during his PhD. thesis. He has performed accurate simulations of the boundary layer than can be seen as a reference to compare with. The results are showed in Figure 4.4, where the line corresponds to R. Olivanti work and the points represent the results of the present simulations. The overall evolution of the curve

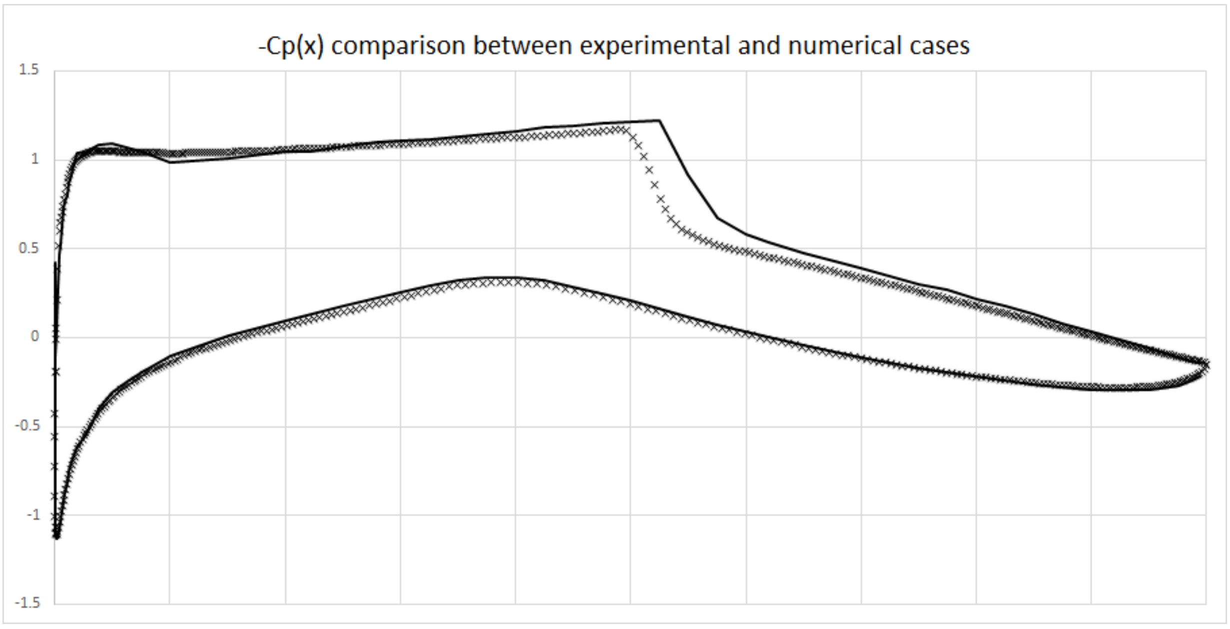

Figure 4.2: Comparison of the distribution of pressure coefficient over the airfoil between experimental and numerical solutions

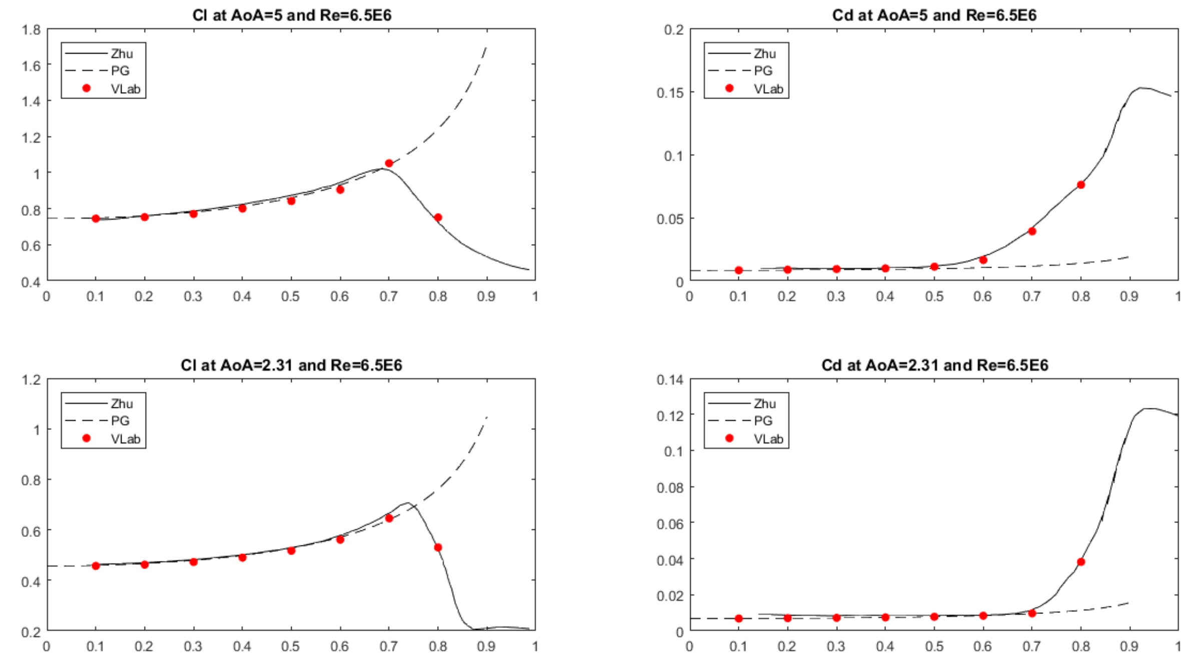

Figure 4.3: Comparison of the results with Zhu's numerical ones and the Prandtl-Glauert transformation

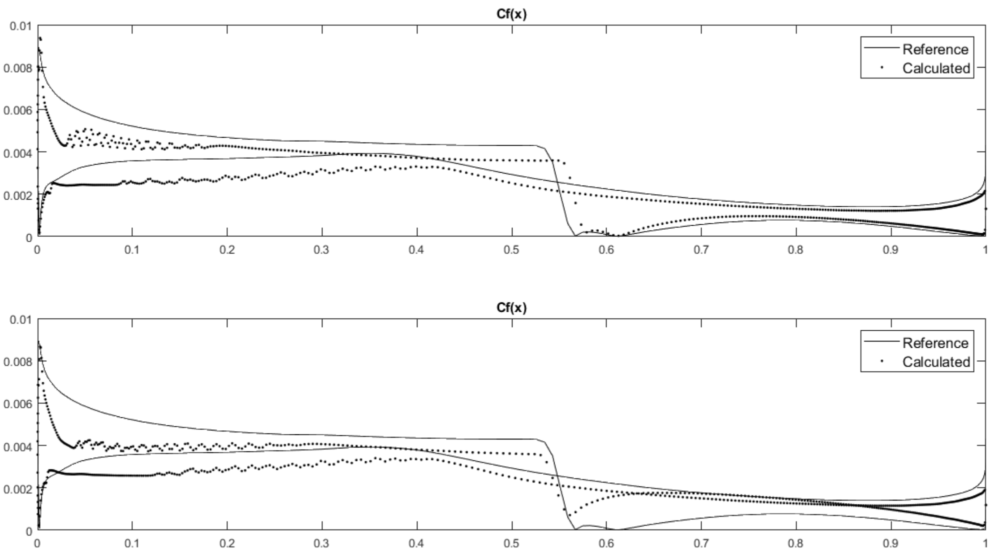

Figure 4.4: Friction coefficient distribution

is correct, presenting a zero value in the stagnation point followed by a peak in the leading edge. The descent caused by the shock wave in the suction side is correctly calculated and its position is not far from the reference one. However, the calculated simulations present oscillations in both sides of the leading part of the airfoil that do not appear in Olivanti's simulations and that have no physical meaning, inducing some errors in the calculation of the boundary layer.

4.2.1 Final database

The results of all the simulations are shown in Figures 4.5 and 4.6, where the horizontal axis refers to the Mach number and the vertical one to the angle of attack. The range of the inlet Mach number has been set between 0 and 0.9 and the incidence between 0 and 10 degrees, as further cases are more difficult to converge and add no relevant information for the considered problem. To built the random database, firstly 21 random values of the incidence have been chosen and, for each one of them, 50 random Mach numbers have also been selected. The Mach numbers are different for each incidence. All these simulations are shown in Figure 4.5, where the green dots correspond to cases that have converged and the red crosses represent the cases that have diverged.

A different database has been created using the so-called Clenshaw-Curtis distribution [9], based on equation 4.1 and using 33 points on each direction. The results are shown in Figure 4.6, where the red crosses correspond to cases that diverged. It can be seen that

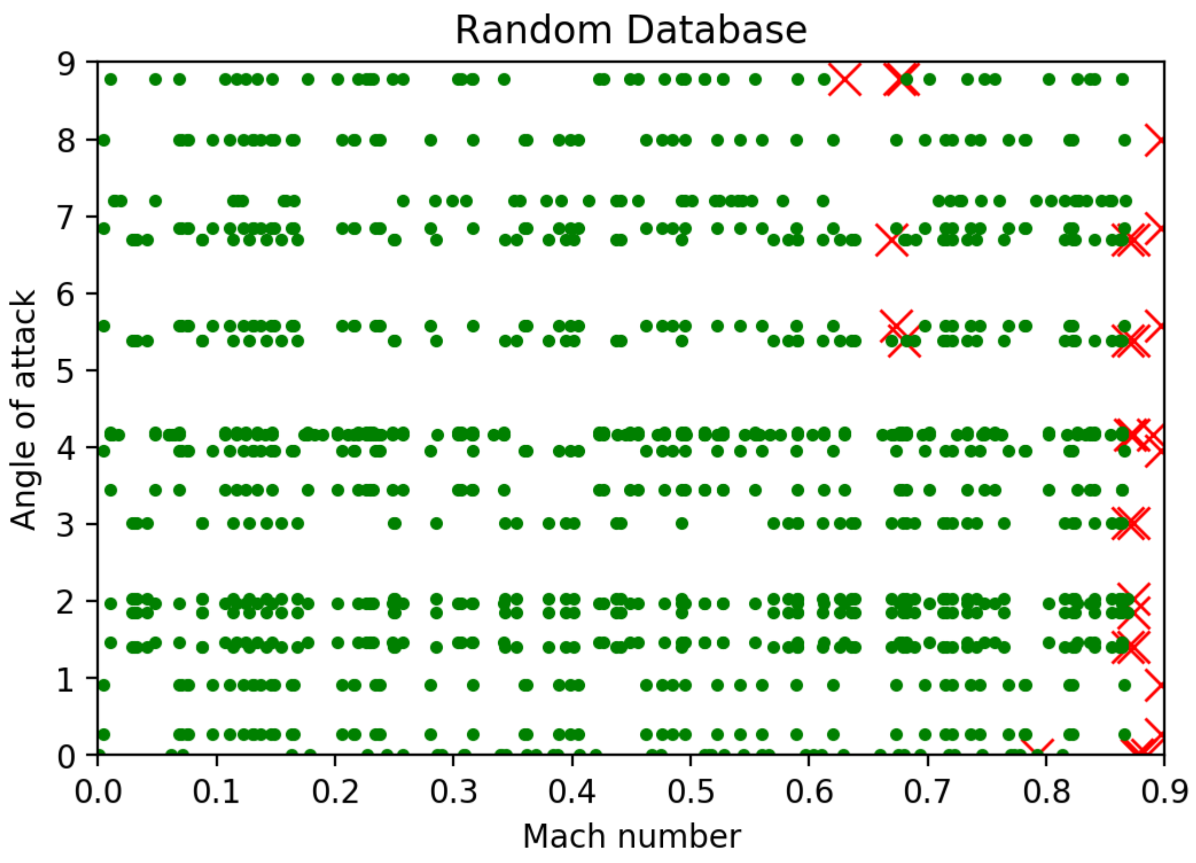

Figure 4.5: All the cases of the database. The red crosses correspond to cases that diverged

most part of these points correspond to the higher values of the incident velocity, for Mach close to 0.875. In addition, there is a zone with Mach between 0.6 and 0.7 and incidence superior to 5 degrees that also diverges. These cases correspond to situations where the flow is very unstable, even non-stationary, which makes the solutions harder to converge. This is the same region that appears in the random database. The divergence of these cases is due to the change of the mesh from structured to unstructured and the solutions can be obtained by refining the mesh. In the random database, a thinner mesh has been used to correct most of the points, whereas this correction has not been carried out for the other database.

It can be seen how the majority of the cases that have diverged are the ones with the highest velocities, independently of the incidence. This is caused by the complexity of the flow in the trailing edge of the airfoil, where there are two shock waves, one in both the suction and pressure side of the airfoil, that detach the boundary layer at both sides of the airfoil, interacting one to another. A more refined mesh in this area is required to make these cases converge but, for the purpose of this work, it is not necessary to solve them.

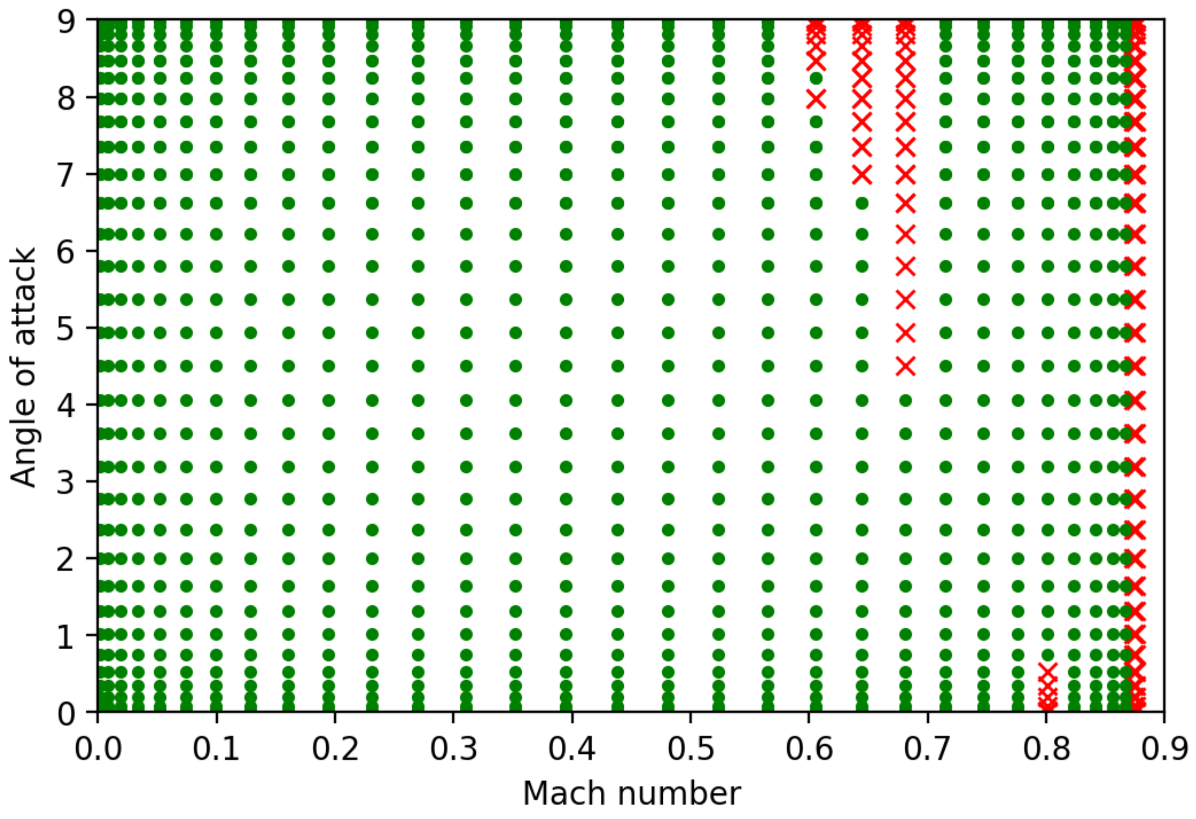

Figure 4.6: All the cases of the database. The red crosses correspond to cases that diverged

4.3 Creation of the model

Once the databases have been created and validated, a model able to learn the desired transformation needs to be chosen. To do so, MultiScale Net, a complex architecture made of several convolutional layers, has been used, as it has proved to provide good results, as in [33] or in the PhD thesis of A. Alguacil and E. Ajuria. Then, some of the data is used to train the network and, the remaining, to quantify the prediction's accuracy of the model, after optimization of the hyper-parameters of the network.

This model, as any other CNN based model, can only take structured rectangular grids as inputs in order to perform the convolutions. This implies that the solutions provided by Fluent must be interpolated into a squared grid, typically 64x64, committing already an error. Interpolations are drawn as red arrows in Figure 4.7, which displays all the operations performed in the model. The model provides outputs in a grid with the same resolution than the input. In order to compare the aerodynamic forces over the airfoil, the distribution of pressure on its walls is required. However, the points on which the predictions are made are not on the surface of the airfoil and another interpolation is required to recover the pressure coefficient distribution. There are, then, three different

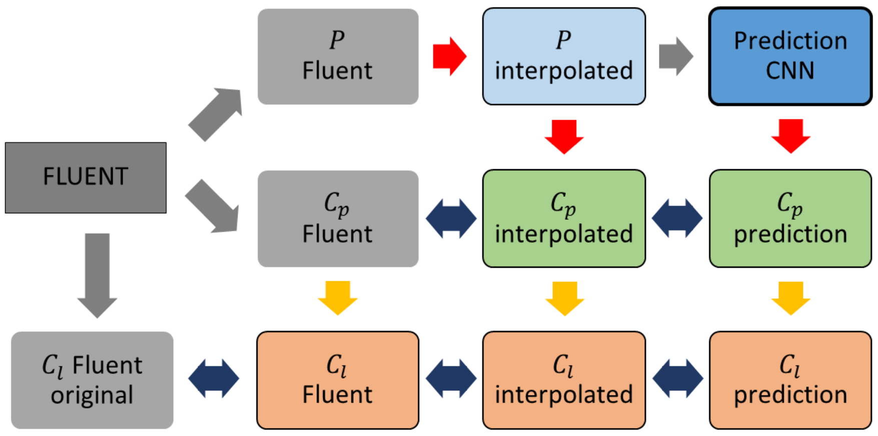

Figure 4.7: Diagram of the methodology followed in this work. The red arrows correspond to interpolations, the yellow ones to integrations and the blue ones represent the comparisons between the elements.

calculations of the Cp distribution that can be compared between them, as expressed by the double-sided blue arrows in Figure 4.7. In addition, an integration of this pressure distribution can be made to obtain the lift coefficient of the airfoil and thus allowing to compare them in a more objective way. The integrations are represented as yellow arrows.

4.3.1 Architecture of the network

The chosen architecture for this work has been the MultiScale Net, developed by M. Mathieu et al. in [16] and showed in Figure 4.8, using a L^2 loss functions. This model uses a mixed architecture, with three parallel branches that share information with each other sequentially. The idea of the model is to learn features at different scales and combine all of them to get a complete model that reproduces more accurately all the information. To do so, a first branch with a Quarter-Scale is computed using 4 2D convolutions with the ReLU activation function (Equation (2.26)). In the end, a bi-linear up-sampling is performed to double up its scale. Then, a second Half-Scale branch is computed, taking the down-sampling of the original data plus the exit of the Quarter-Scale branch as inputs and performing 6 2D convolutions with the ReLU activation function. The procedure is repeated a third time with the Full-Scale data and a bi-linear up-sampling of the exit of the Half-Size branch. With all these operations, a field of the same size as the input data is provided. Notice that the only parameters that need to be trained are the weights defining the filters of the convolutions, represented in grey in Figure 4.8.

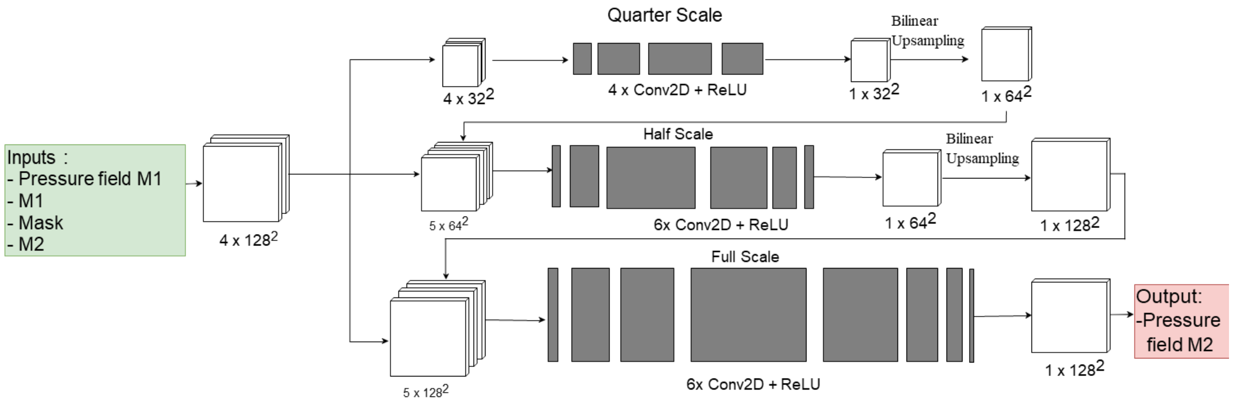

Figure 4.8: Architecture of the MultiScale Net

Hyperparameters of the network

The hyperparameters of a network are the variables that determine the network structure and how a network is trained, like the number of filters or layers, the activation functions or the optimizer used or the numbers of epochs of the training. As the architecture has already been chosen, the hyperparameters in this case consist of the selection of the learning rate and the number of epochs used for training. These variables are set by trial and error based on the evolution of the loss function over the epochs, which is similar to a decreasing exponential function as shown in Figure 4.9. The learning rate has been set to 0.001 for the first 200 epochs, reducing it to its half for the rest of the 400 remaining epochs to raise the precision of the model. The final model has been set as the one that has the minimum value of the loss function for the validation set, composed by a random 20% of the data as explained later in section 4.3.2. The L^2 loss function has been chosen as it gives more importance to the points of maximum and minimum pressure that are crucial for this problem.

4.3.2 Data use for training

Once the database has been created, it needs to be pre-processed before being fed to the network. As a Convolutional Neural Network-based model is being used, its inputs need to be 'images': rectangular grids with the different values of each pixel (in this case, the value of the pressure on each point of the grid). The database provides the solutions in each cell of the mesh and can not be introduced directly into a CNN because the convolutions can not be performed. To solve this problem, an interpolation from the field on the original Fluent mesh and the desired rectangular grid (typically 64x64 or 128x128) must be carried out.

Once the data have the correct dimensions, a decision about which data have to be

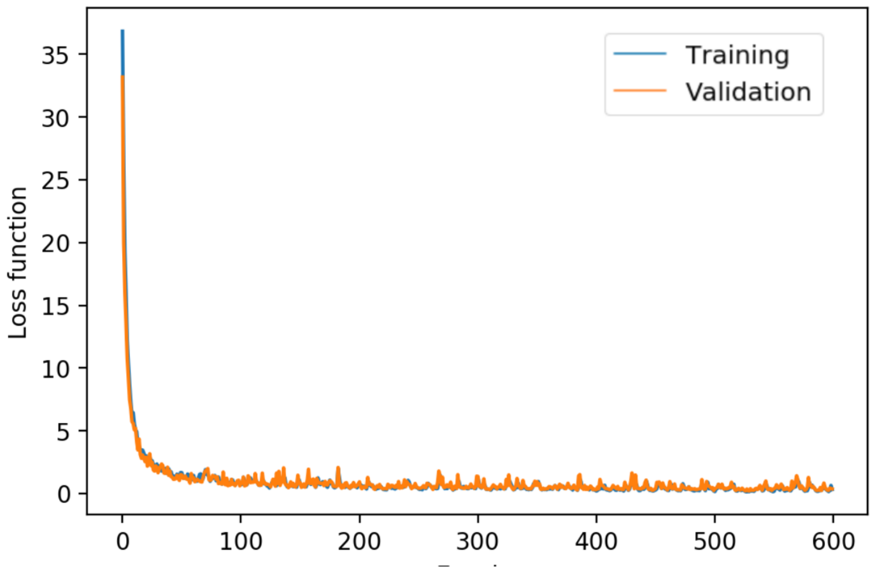

Figure 4.9: Loss function evolution over training epochs

given to the network must be made. In this work, the input data is:

- Pressure field at low Mach: interpolated over the considered grid
- Initial Mach: a matrix with the value of the initial Mach on each point
- Mask: A matrix with 1 in the pixels that are inside the airfoil and 0 elsewhere
- Final Mach: a matrix with the value of the final Mach on each point

The output of the network is going to be the pressure field at the final Mach, higher than the initial one. The inputs and outputs are also shown in Figure 4.8. Some additional variables could be taken into account, like the density or velocity fields, but only solutions with the pressure have been considered. In the future, some of these variables could be added trying to raise the precision of the model.

Now that all the cases are prepared to be fed to the network, it must be decided which cases are given to the network for training and which ones are used for validation and testing purposes, as explained in section 2.2. As a first step, and to see what the network is actually learning, only one angle of attack has been considered for training: 4.17 degrees. From all the couples with this incidence, a random 60% has been used for training and another random 20% for validation. The rest of the cases at 4.17 degrees of incidence, as well as all the cases with different angles of attack, have been used for testing, measuring how good are the predictions of the model. If the model is not able to provide accurate enough solutions for different angles, some additional cases could be added to the training set.

Interpolation

As explained before, inputs of CNN neural networks should be interpolated when arising from unstructured data. The most used grids in computer vision are 64x64, 128x128 and 256x256, but any other resolution could be used like in the ongoing PhD. thesis of E. Ajuria or A. Alguacil.

In addition, to improve the resolution of the picture, only the region closest to the airfoil is taken into account, as the far regions contain no relevant information about the problem. For this work, the region $x \in [-c/4, 5c/4]$ and $y \in [-c/2, c/2]$ is considered, where c is the considered chord of the case, that changes with the velocity as the Reynolds number remains constant. The interpolation has been carried out with the 'griddata' function from Python 3.7, using a cubic spline, as it provides the most accurate results. Besides, using cubic splines implies that the interpolation values are C1 class, with the interpolated values and its derivatives being continuous.

Choosing the correct resolution of the grid is not a trivial question: if a small one is taken, the calculation time is going to be smaller but the precision of the calculation is also going to be reduced, whereas choosing a high-resolution grid acts in the opposite way. To quantify the error made with the interpolation, the curves of Cp(x) over the airfoil for two different regimes are showed on Figures 4.10 and 4.11. It can be seen how the larger the number of points on the grid, the closer to the real Cp, calculated on the wall by Fluent. In addition, the information lost due to the interpolation is more important in the leading and trailing edges, where the pressure gradients are more important, leading to low accurate aerodynamic coefficients for the lift, drag and momentum. However, using a grid with twice the points on each direction multiplies by 4 the total amount of points for each case and increases the computational capacities required. So, in the end, a compromise solution needs to be reached.

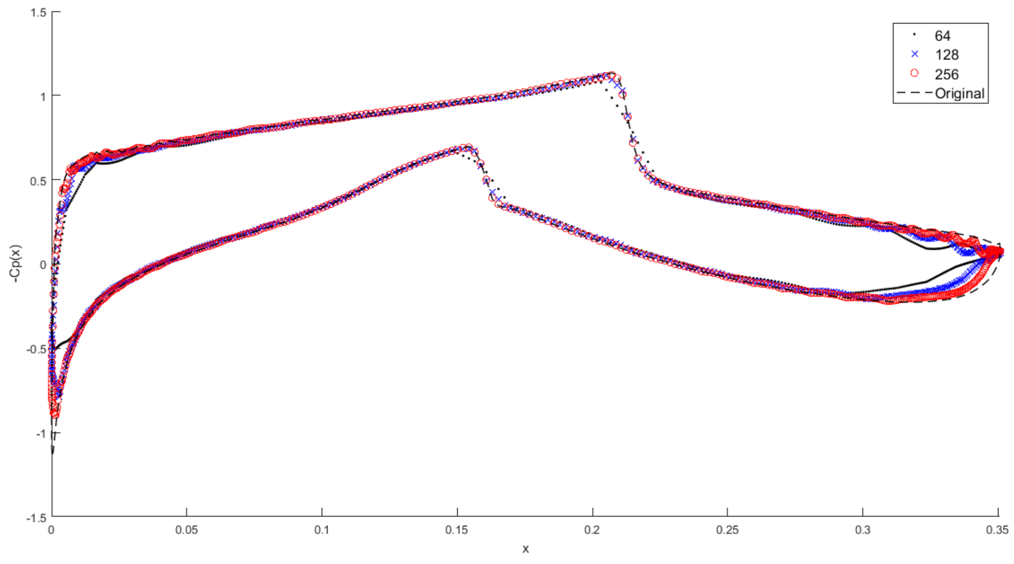

Figure 4.10: Comparison of Cp distribution calculated by Fluent and the one interpolated with three different resolutions for $M = 0.823$

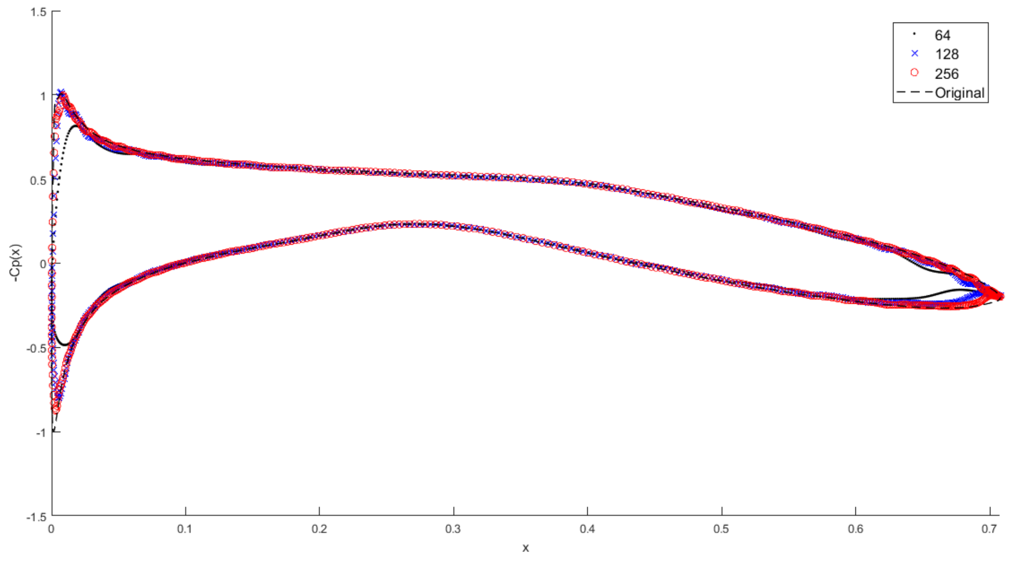

Figure 4.11: Comparison of Cp distribution calculated by Fluent and the one interpolated with three different resolutions for $M = 0.522$

4.3.3 Results of the Model

The model considered is trained with a random 60% of the couples with an incidence of 4.17°. The evolution of the loss function during training can be seen in Figure 4.9, where no evidence of overfitting or underfitting appears, so the training can be considered as good. Now, what the network has learned must be checked in order to validate its predicting capabilities. The output of the network is always a square grid (64x64 for example) representing the value of the pressure on each one of these points. To quantify the accuracy of the predictions, one could simply measure the difference of each pixel with respect to the pixels obtained by interpolating the pressure field calculated with Fluent. However, this method gives no importance to the regions near the wall, which are the most important ones in order to get the aerodynamic coefficients. Instead of doing this, the pressure coefficient distribution on the wall has been compared using the MAE and the RMSE.

Three different ways of calculating the pressure coefficient distribution have to be considered. Firstly, the Fluent simulations of the database provide directly these curves as the values of the pressure on the surface of the airfoil are known: this curve is the Ground Truth. Secondly, a Cp distribution can be obtained by interpolating the values of the pressure on the grid (previously interpolated from the Fluent mesh) over the wall of the airfoil. This Cp curve will be less accurate due to the error made by interpolating but will allow the quantification of this difference. Finally, the predicted values of the network, in a grid format, can similarly be interpolated over the surface, obtaining the new Cp predicted distribution.

Results for the training angle

As a first model, a resolution of 64x64 has been used as it constitutes a good compromise between accuracy and computational efficiency. A random 20% of the cases was reserved to test the performance of the model. However, a distinction has to be made between two different test situations. On the one hand, to create the test set, a random 20% of the couples has been chosen. These pairs have never been seen by the model but both the input and the output pressure fields have already been used independently in different couples of the training set. Some examples of the results of these cases are shown in Figures 4.12 and 4.13, where the upper left picture corresponds to the initial field, both pictures on the right side correspond to the prediction of the network and the Ground Truth provided by the interpolation from Fluent over a 64x64 grid. The lower left picture corresponds to the three cases of Cp explained above. On the other hand, some additional couples have been calculated in order to observe how the model predicts on completely new cases.

To measure quantitatively the error of the model and identify how much of it is due to the network and how much is due to the interpolation, the MAE and the RMSE of

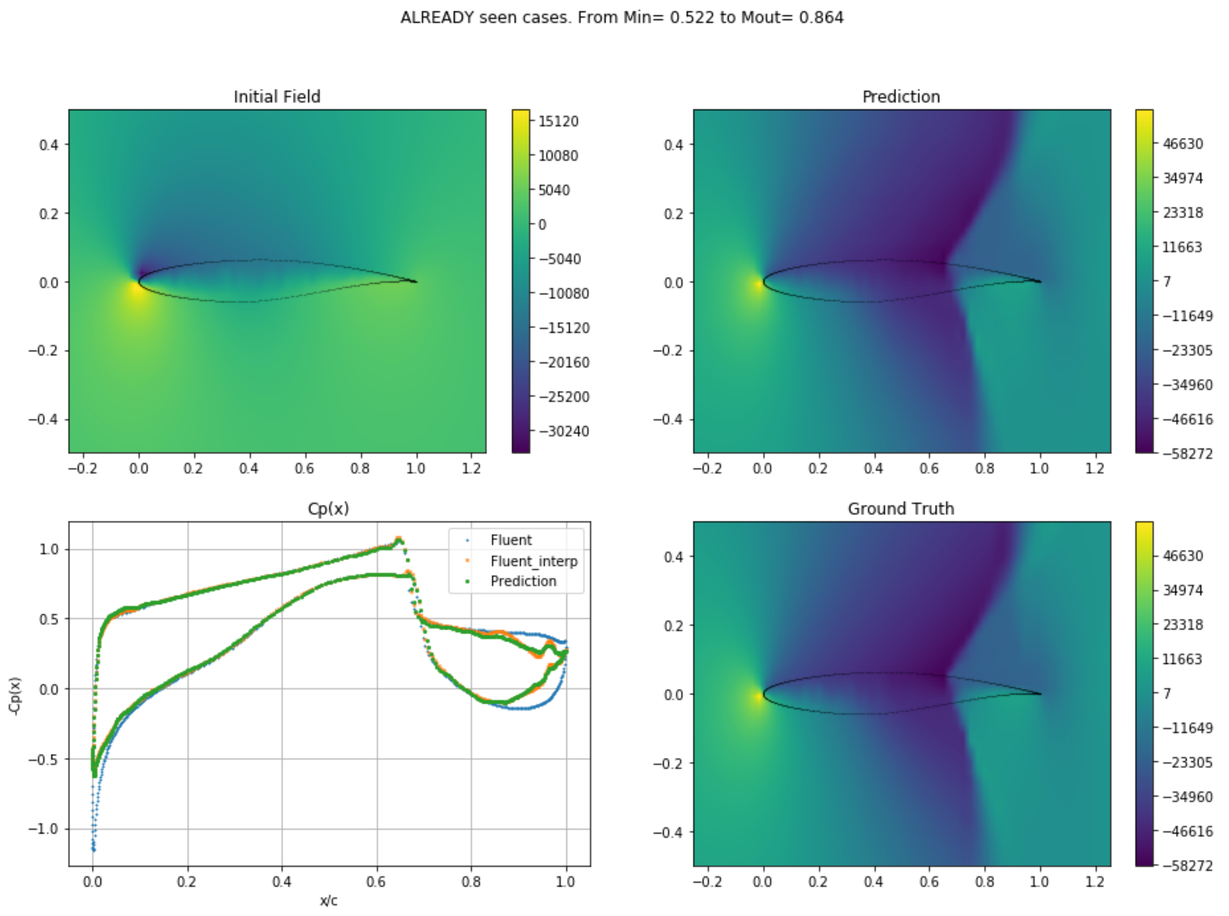

Figure 4.12: Example of a prediction of the model for 4.17 degrees of incidence

the pressure coefficient on the surface of the airfoil for each one of the test cases have been calculated. The results are represented in Figures 4.15 and 4.16 respectively. The colors represent the value of the error and each red point corresponds to one single couple, consisting of transforming from M1 (x-axis) to M2 (y-axis), being $M2 > M1$. The upper left picture corresponds to the error made by interpolating the solution on the Fluent mesh to the grid of 64x64 and interpolating again over the surface of the airfoil. The upper right one represents the difference between the original Fluent distribution and the prediction of the network, in 64x64 resolution, interpolated over the wall. On the lower left one, the difference between the prediction and the Fluent interpolated over a grid of 64x64, both interpolated over the airfoil, is depicted. Finally, the lower right one represents the three Cp distributions for the worst case in the figure of the MAE and the worst relevant case (with a predicted field at Mach number higher than 0.2) in the RMSE one.

From these figures, it can be deduced that the main contribution to the error is caused by the interpolation, as the difference between the prediction and the Fluent interpolated is much smaller than the error of the interpolation. In addition, it has to be noticed that only

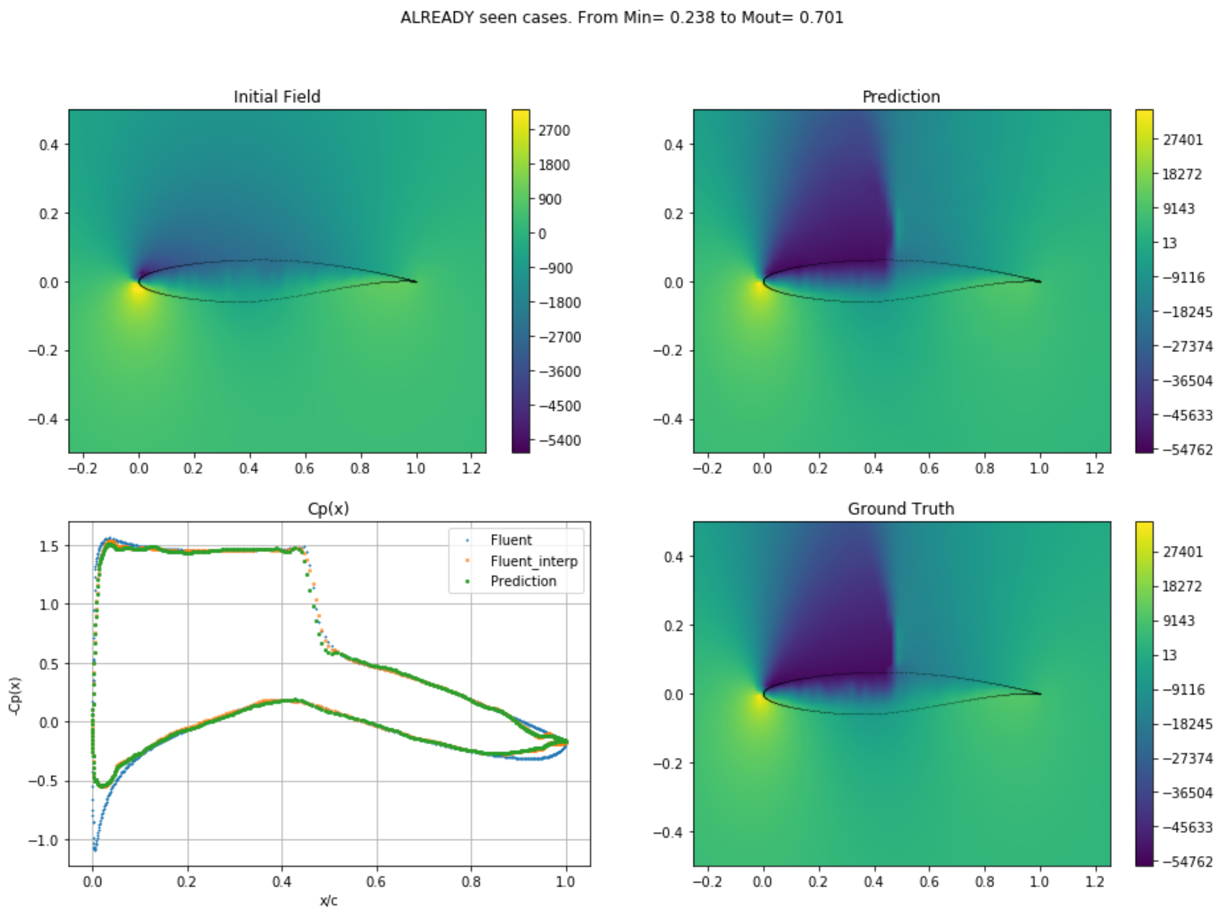

Figure 4.13: Example of a prediction of the model for 4.17 degrees of incidence

100 contours have been plotted on each picture, appearing blank zones that correspond to cases whose error is bigger than the set limit (0.1 for the MAE and 0.05 for the RMSE).

A similar procedure can be followed with the completely new cases that have never been seen by the network during training. The results do not vary much with respect to the previous ones which indicates that the network is not just memorizing the training examples but actually learning the transformation. The results are shown in Figures 4.17 and 4.18 where a similar conclusion can be made: the main cause of the error is the interpolation and not the network itself. This can be observed in the pressure coefficient distribution of the worst cases that shows how the interpolation is not able to capture the peaks on the leading edge.

In addition, the results have been compared with the two most used similarity rules: Prandtl-Glauert (Equation (1.24)) and Karman-Tsien (Equation (1.25)). The results, showed in figure (4.14), display how the similarity rules get less and less accurate when the Mach raises whereas the created model improves its precision with higher Mach numbers.

The created model is more accurate than the similarity rules for Mach numbers higher than 0.5 approximately while remaining a little bit worst for lower speeds (always having in mind that it does not work for extremely low ones). It is worth mentioning that for this particular case the Prandtl-Glauert transformation provides better results than Karman-Tsien, when the opposite behavior is normally obtained. However, the model is not able to predict accurately the low-Mach numbers fields as showed in the upper left point that appears in Figure 4.14.

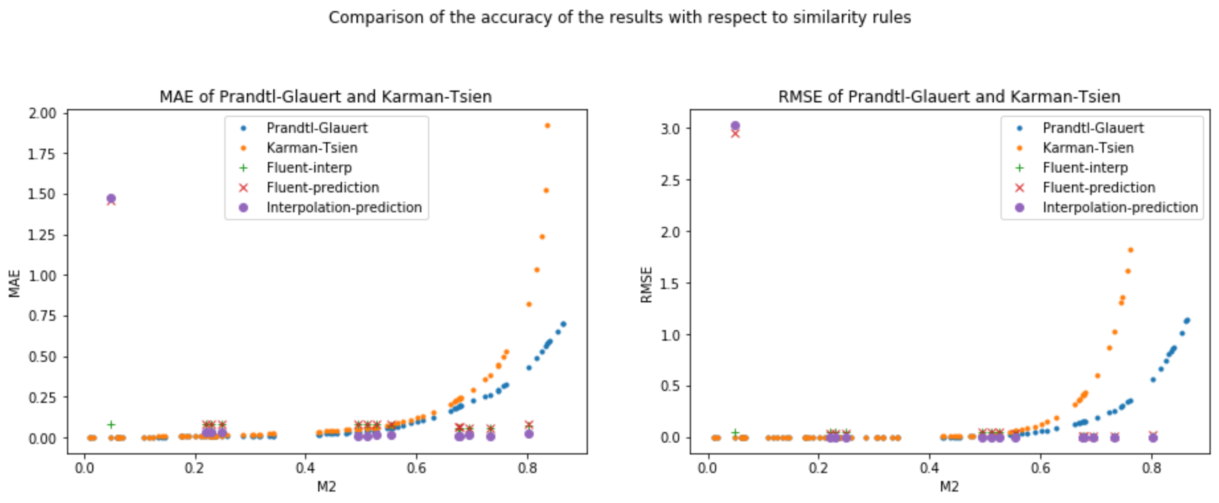

Figure 4.14: MAE and RMSE of the predictions and the similarity rules

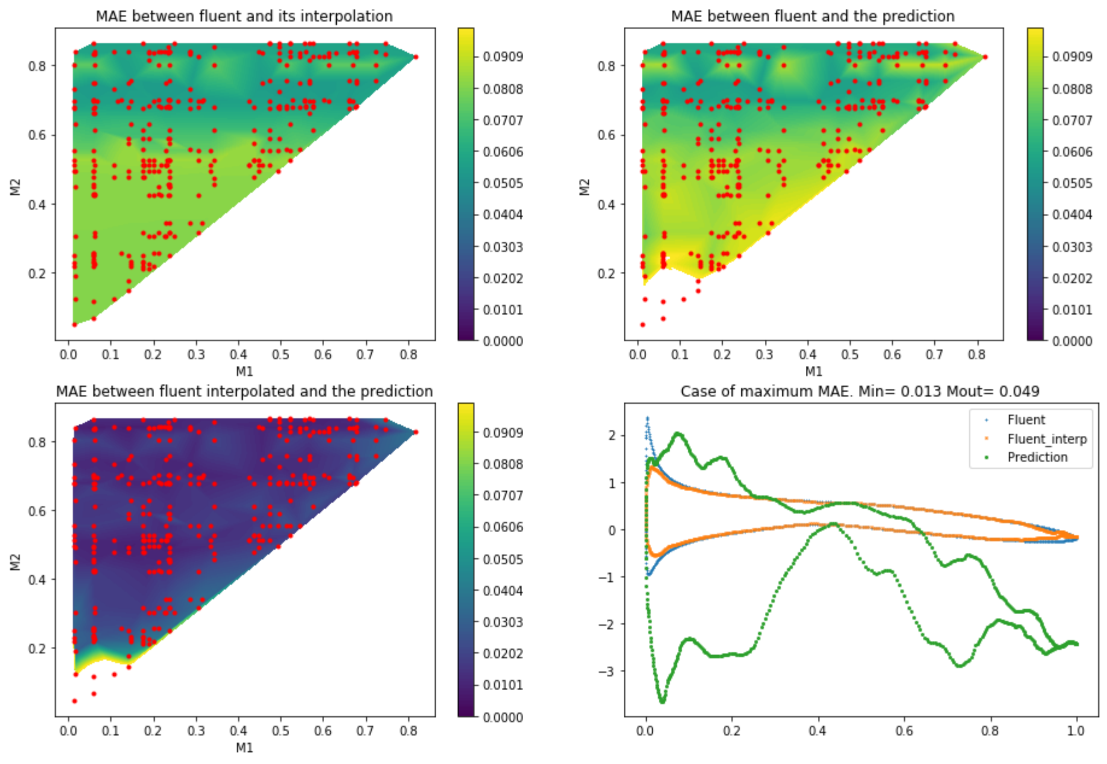

Figure 4.15: MAE of the test set. Only 100 contours from 0 to 0.1 have been plotted

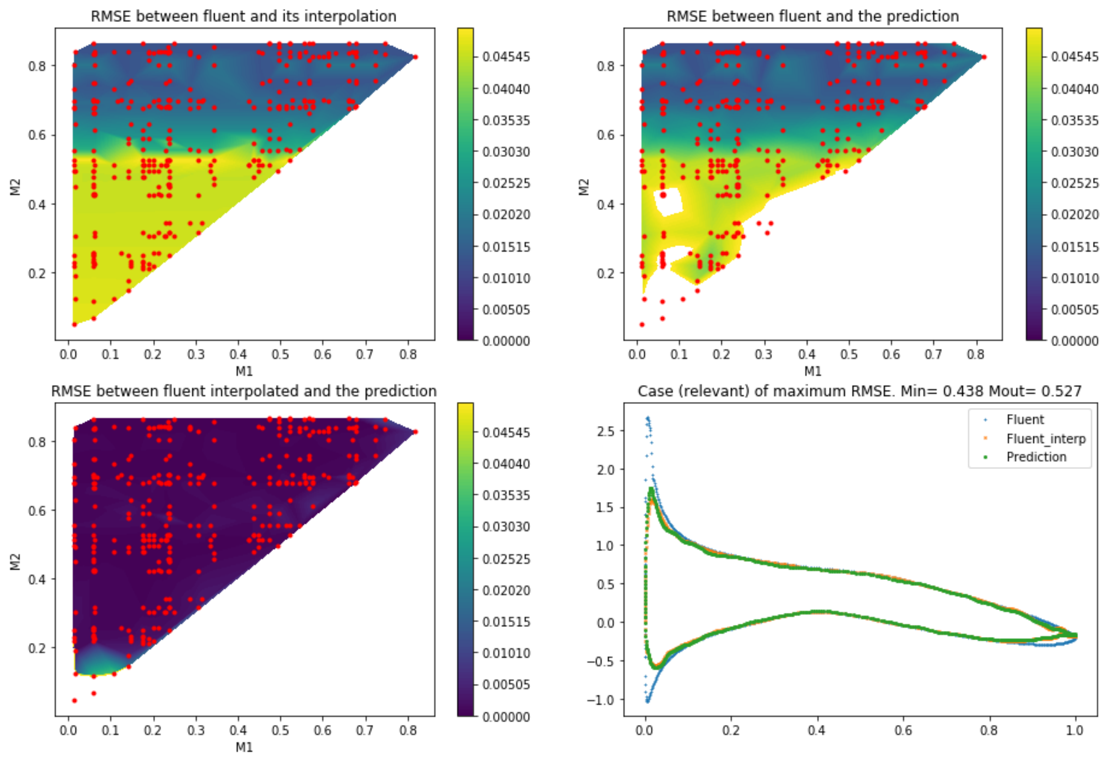

Figure 4.16: RMSE of the test set. Only 100 contours from 0 to 0.05 have been plotted

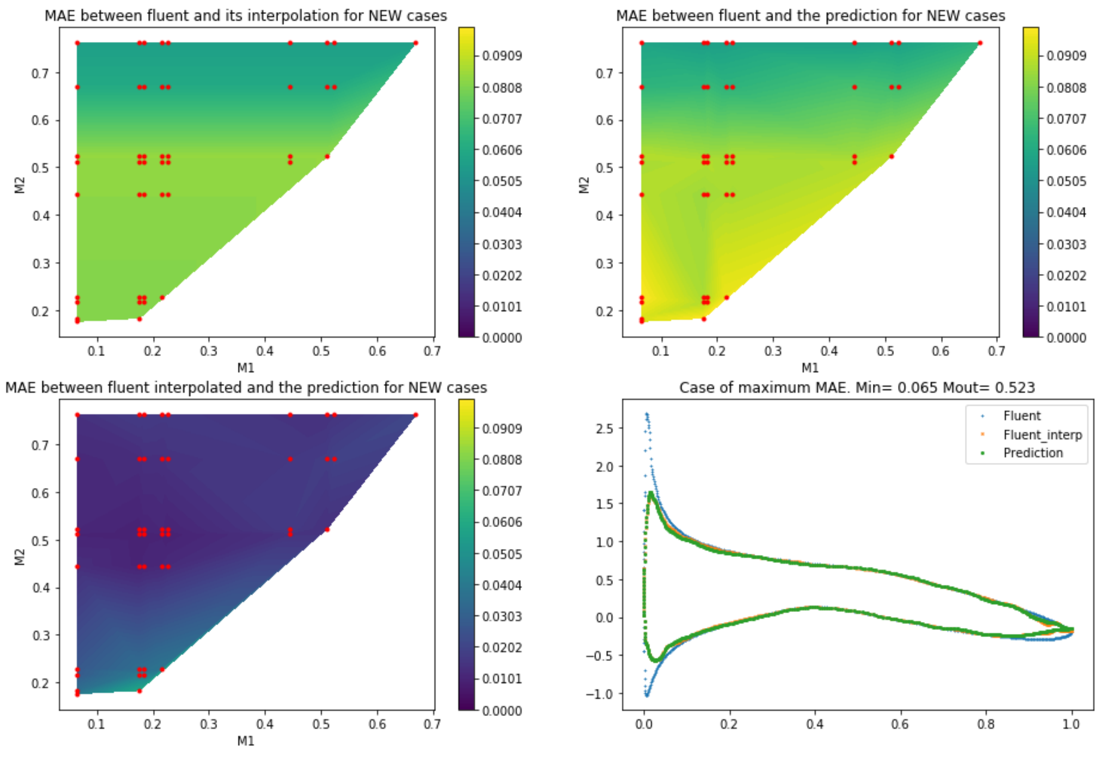

Figure 4.17: MAE for completely new cases. Only 100 contours from 0 to 0.1 have been plotted

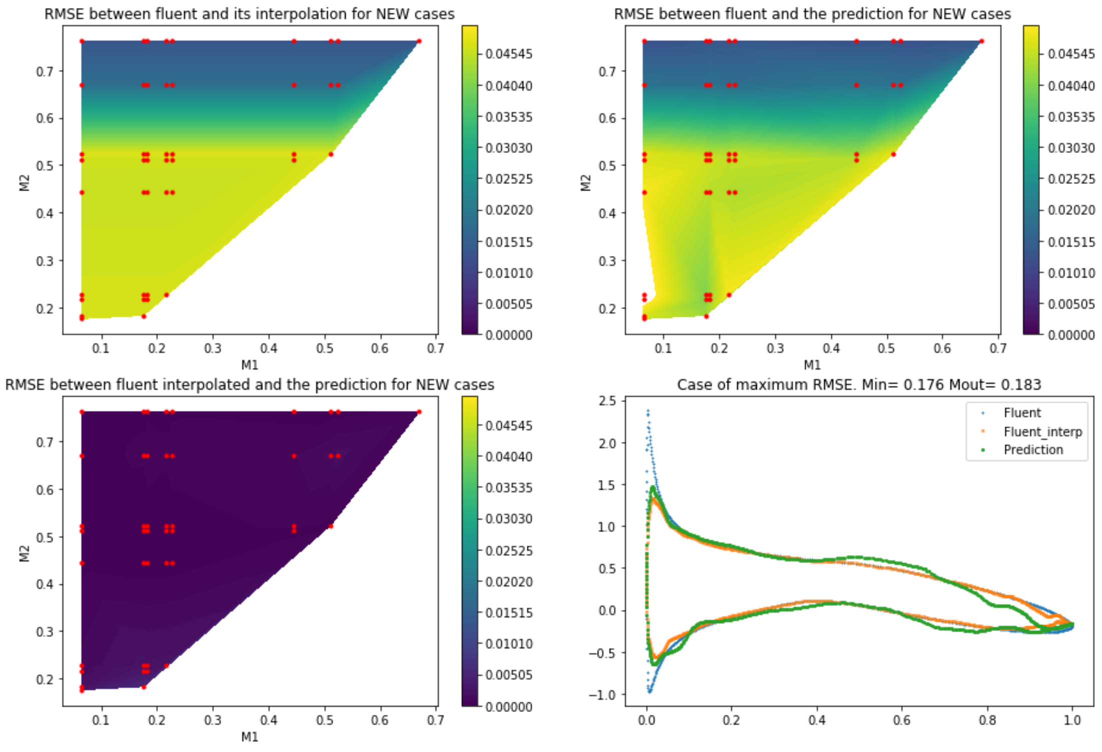

Figure 4.18: RMSE for completely new cases. Only 100 contours from 0 to 0.05 have been plotted

Results for different resolutions

As demonstrated before, the error is mainly caused by the interpolation from the Fluent mesh to the 64x64 grid. Trying to reduce this error, the easier thing to do is to raise the resolution of this grid up to 128x128 or 256x256. The MultiScale model used for this work is independent of the resolution of the data: it provides an output with the same size as the given input. Therefore, the trained network with 64x64 can be used to predict the pressure fields of higher resolution just by giving input fields with higher resolutions. This procedure has been done for some cases with 128x128 and 256x256 resolution and one of the results is depicted in Figure 4.19. It can be seen how the results do not represent the physics of the problem and this method does not improve the accuracy of the model. In addition, Figure 4.20 shows the MAE and the RMSE for the different resolutions, probing that the error is bigger than the one of the original 64x64 picture, thus invalidating this method. This behavior has also been found for other authors, like in [24].

Even though the exact reason why this decay of performance happens is not fully understood yet, the main ideas suggest that it is caused by the difference of the gradients produced when changing the resolution. A difference of 100 units between two adjoining pixels in a given resolution would become 50 if the number of pixels in this direction is doubled. The filters of the network have been trained to detect gradients of 100 and will not work as well for other gradients, thus providing worst predictions, as a similar situation happens in the output of the network.

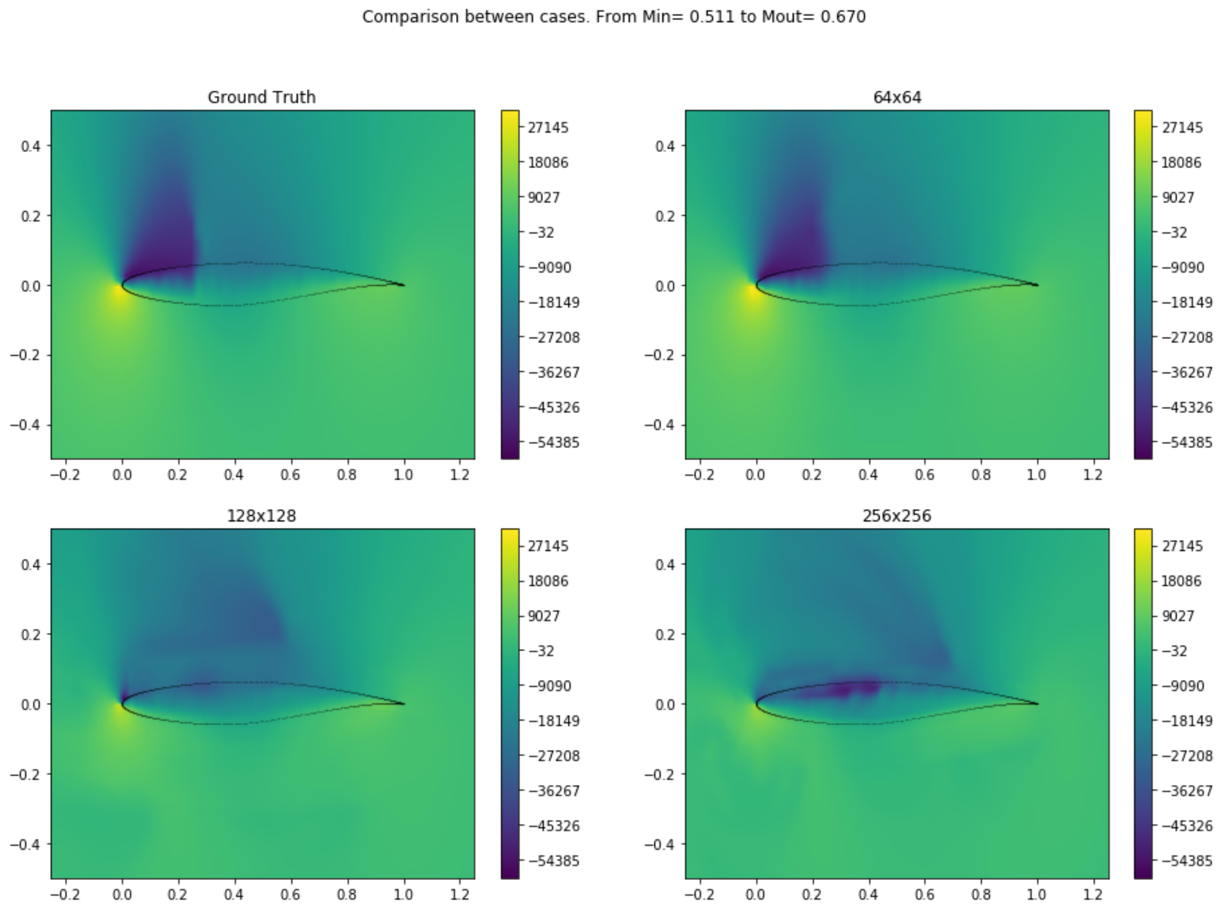

Figure 4.19: Comparison between different resolutions

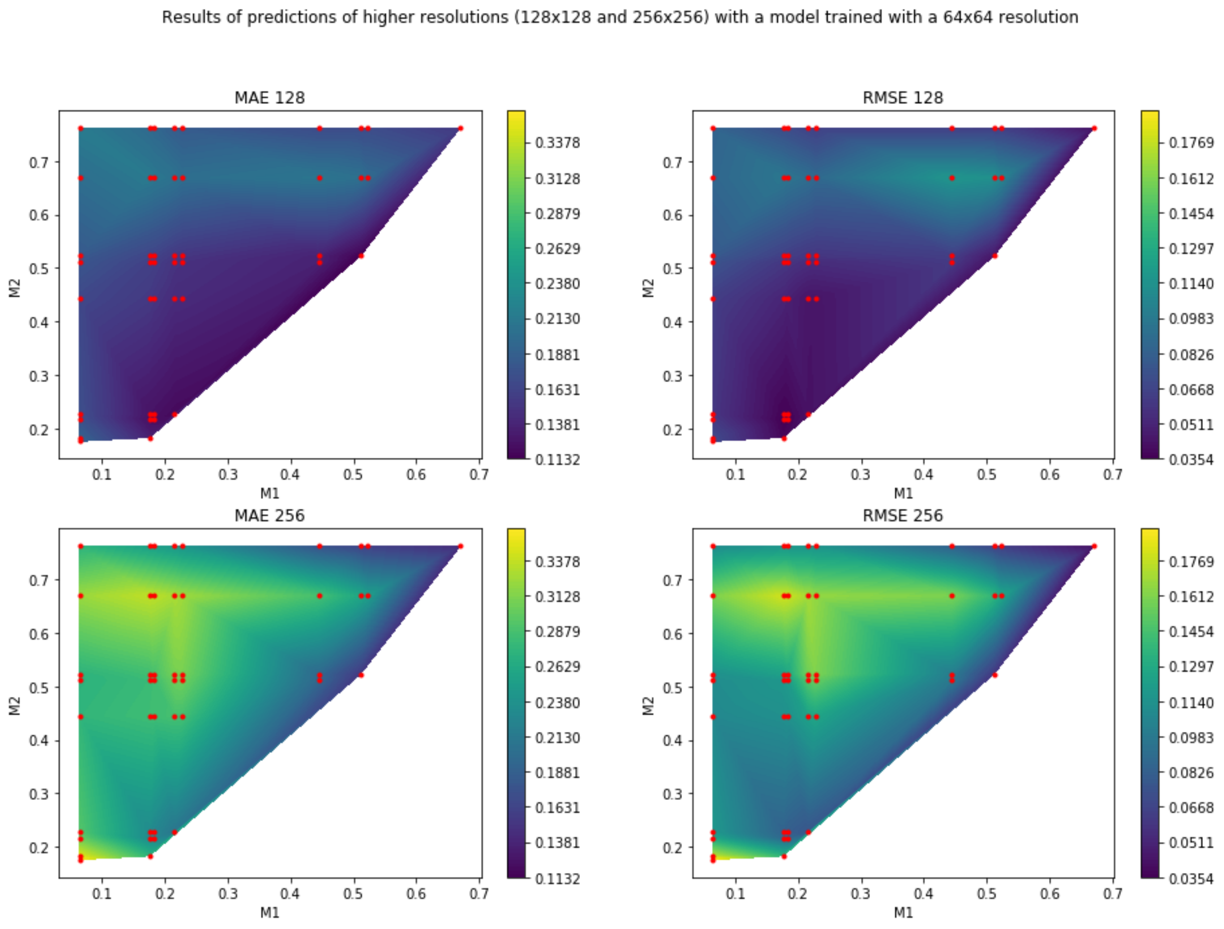

Figure 4.20: MAE and RMSE of the predictions made with 128x128 and 256x256 resolutions by a network trained with 64x64 resolution data

Results for different angles

As proved in the previous section, the results for 4.17 degrees, the same angle that has been used for the training, are very accurate. However, the main interest of this work is creating a model that works correctly for any angle of attack. Before creating a new model trained with different incidences, where a new variable will be added to the inputs, increasing around 25% of the computational requirements, the extrapolation capability of the current model for different incidences has been verified.

To do so, 10 different Mach velocities randomly picked have been used for each one of the 20 angles of the random database. For each angle 45 couples can be created, counting with a total of 900 cases. This will allow to see how the error changes when the angle distances from 4.17 degrees and the differences between the different transformations on each case (from high or low Mach to higher Mach). The results of the MAE and the RMSE of the pressure coefficient distribution are showed in Figures 4.21 and 4.22 where the distribution of the subfigures is the same as in the previous ones.

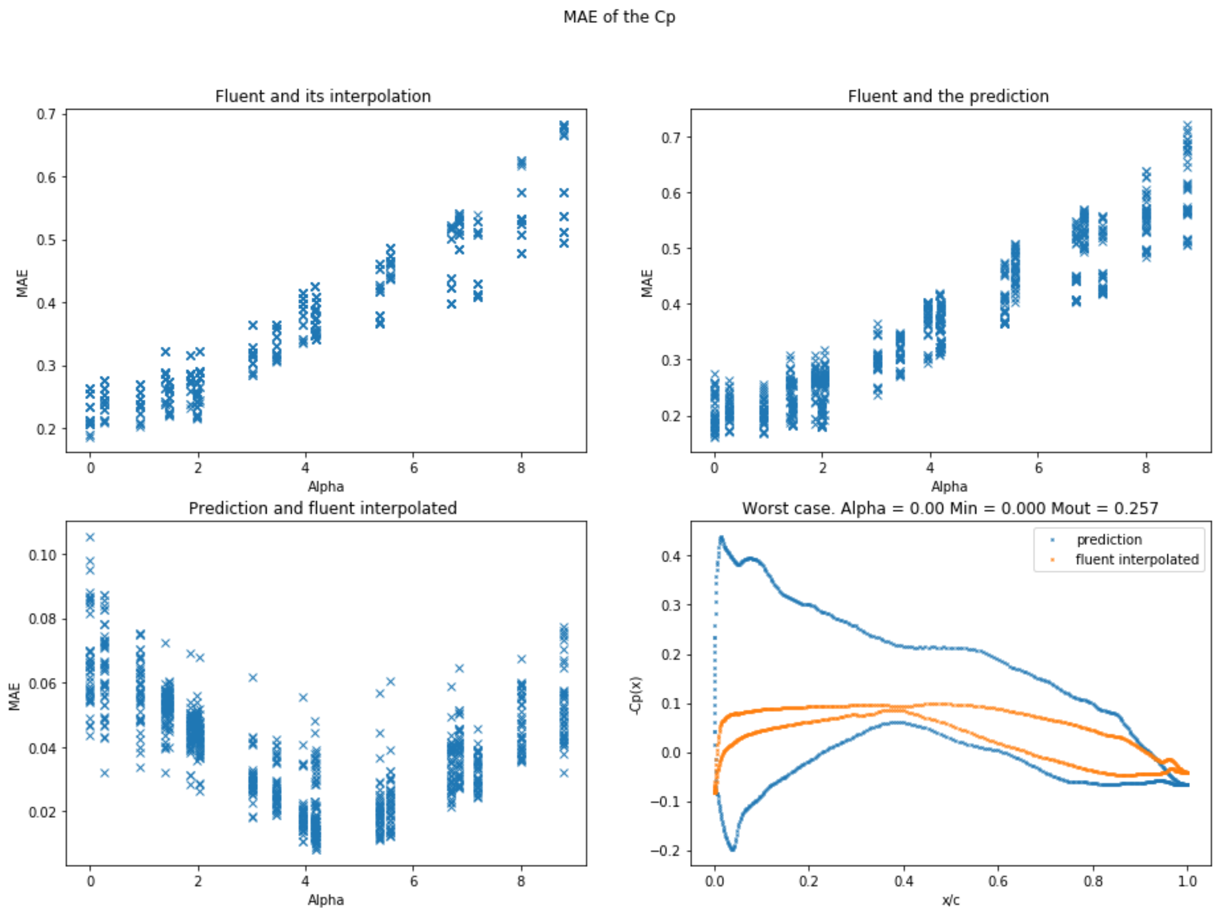

Figure 4.21: MAE for different incidences

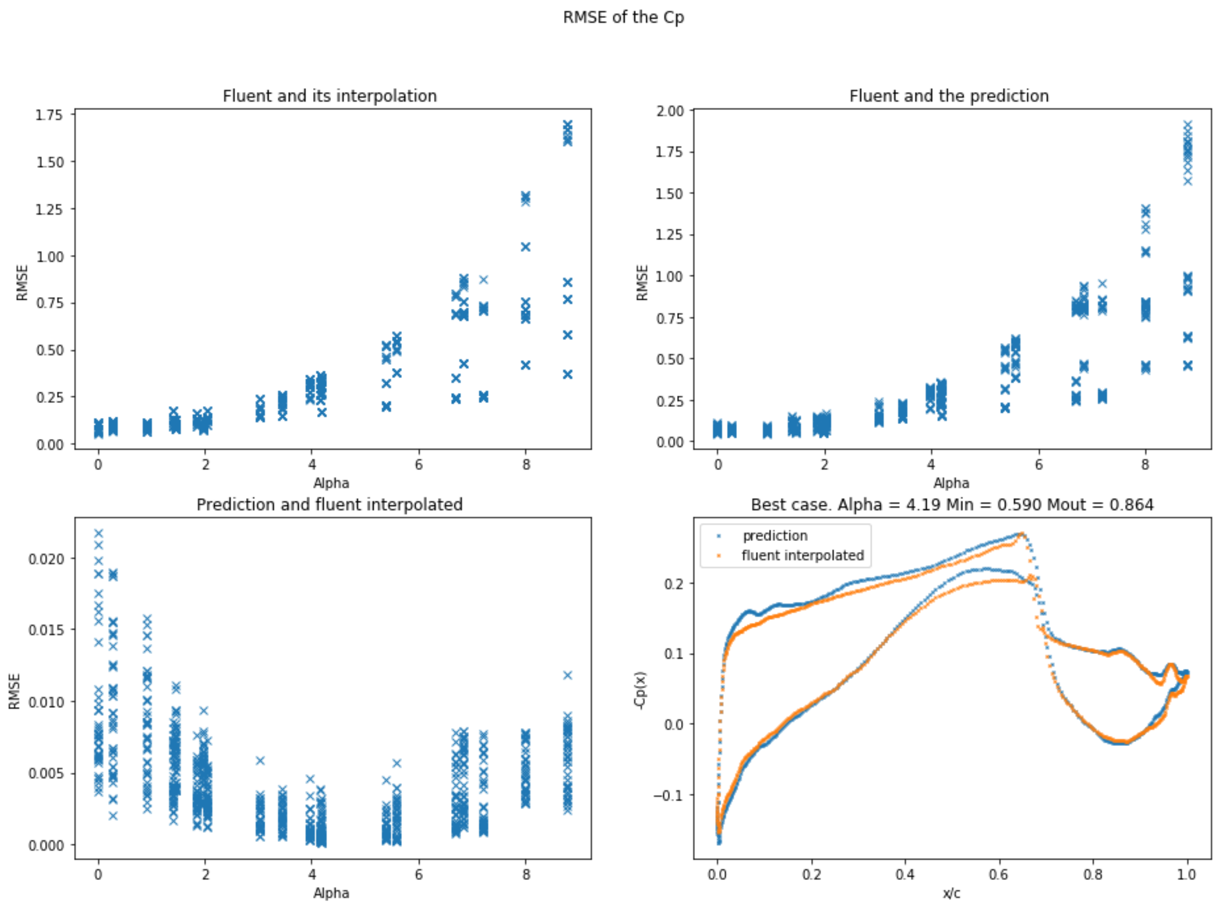

Figure 4.22: RMSE for different incidences

It has to be noticed that the MAE and the RMSE measure the absolute error, not the relative one. That implies that both variables are going to raise with the incidence of the considered case as the limit values of the pressure raise too (the same reasoning can be made for the incidence velocity but its influence is minor). This explains why the upper figures grow with the incidence instead of presenting a minimum around 4.17 degrees.

Comparison of obtained Lift Coefficients

The lift coefficient has also been calculated for each calculated cases by the integration of the pressure over the surface of the airfoil, obtaining the results shown in Figure 4.23, where the relative errors with respect to the value of the Cl provided by Fluent are depicted. The upper left picture shows the error made solely by the interpolation, integrating the pressure distribution provided by Fluent, which is less than 1% for the most part of the cases. However, some cases present higher errors, especially those where two shock waves appear, like the one shown in the lower right picture. If the input of the network is interpolated into a 64x64 grid, re-interpolated over the surface of the airfoil and then integrated, the total error committed is around the 10%, as shown in the lower left picture. The total error of the model, obtained from interpolating the output of the network and integrating the results, is depicted in the upper right picture. It can be seen how it presents a minimum value around 4.17 degrees as expected, with a total error of less than 25%. However, this curve expresses how the larger angles are predicted better than the lower ones, whose predictions are very far from the real solutions, as showed in the remaining picture.

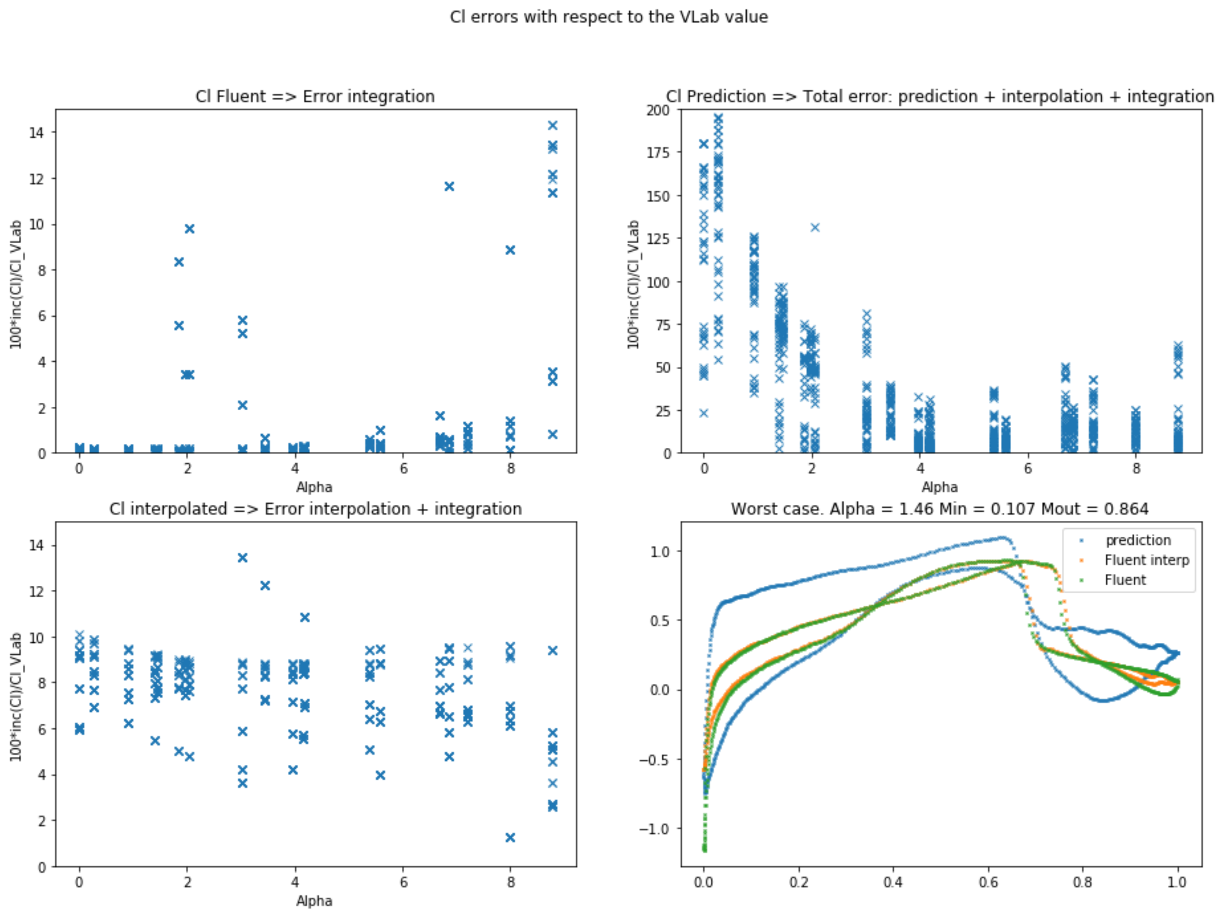

Figure 4.23: Relative error of the different lift coefficients with respect to the Ground Truth

4.4 Implementation of the model in a LBM solver

The previous results show that the created model is able to accurately predict the transformations from incompressible to compressible flow. The idea now is to perform these transformations to more precise incompressible solutions, like those provided by the Lattice Boltzmann method, aiming to obtain more accurate solutions for the compressible field. To do so, simple subsonic cases for the same airfoil (RAE 2822) need to be calculated using the LBM. Then, the field is to be interpolated to the desired grid in order to be fed as an input of the network, obtaining the compressible solutions on this same grid. From this information, one can interpolate the values on the airfoil surface to obtain the pressure coefficient distribution and integrate it to obtain the aerodynamic coefficients. This part of the work is to be made in the future as the computational time required to perform the LBM simulations is very high and has not been done by the time this report was written.

Conclusions and Future Work

Conclusions

Two databases have been created using Fluent simulations and validated comparing the obtained results with numerical and experimental data. Then, this database has been pre-processed, creating couples of pressure fields from low to high Mach numbers, and interpolating the results on the mesh used in the simulations over a 64×64, 128×128 or 256×256 grid, depending on the cases. This interpolation has been proved to be responsible for the majority of the error, obtaining the best results with cubic splines.

A MultiScale Net architecture, composed of several convolutional layers, has been trained solely with the couples at 4.17 degrees of incidence in a 64×64 grid. The results for extremely low Mach numbers (under 0.1) are not good but they are not relevant for the considered project. However, for the rest of velocities, the results are very accurate, outperforming both the Prandtl-Glauert and the Karman-Tsien similarity rules for $M > 0.5$, while remaining similar for lower velocities.

Then, trying to reduce the error caused by the interpolation, this same model has been used to predict the fields with higher resolutions: 128×128 and 256×256. The results are worst than the previous ones and this error raises with the resolution.

To check the robustness of this model, fields at different angles of attack, from 0 to 10, have been predicted. The results at angles close to $4.17°$, used for the training, are still good but the precision decays as the angles distance from 4.17. In addition, the results for small angles are worst than the ones at high incidence.

Future work

Solutions with the LBM at low-Mach number need to be calculated and introduced in the model to check the accuracy of the transformations. A model trained with the values taken from the Clenshaw-Curtis distribution needs to be created and its results compared with the ones provided by the random distribution to probe the independence of the model to the input data.

Then, some additional models need to be created to overcome the interpolation error. For example, a model trained with higher resolutions fields can be created, by the computation capabilities required would also raise. Following [3], an ANN could also be added at the end of the model as a substitute for the interpolation, providing directly the pressure distribution over the airfoil. This network could be trained independently that the previous one (Transfer Learning) or both could be retrained jointly. The latter, according to the bibliography, is meant to provide better results, but both solutions must be checked out.

To improve the error caused for incidences far from the training one, an additional input channel with the angle of attack of each case could also be considered and some other angles could also be used in the training.

Bibliography

[1] Palabos. `http://www.palabos.org/`.

[2] J. Anderson. *Fundamentals of Aerodynamics.* McGraw-Hill Education, 2010.

[3] P. Baqué, E. Remelli, F. Fleuret, and P. Fua. Geodesic convolutional shape optimization. *CoRR*, abs/1802.04016, 2018.

[4] M. Campbell, A. Hoane, and F. hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[5] P. Eliasson. Solutions to the Navier-Stokes equations using a k-epsilon turbulence model. *NASA STI/Recon Technical Report N*, 89, Apr. 1988.

[6] D. C. Forbes, G. J. Page, M. A. Passmore, and A. Gaylard. A study of computational methods for wake structure and base pressure prediction of a generic suv model with fixed and rotating wheels. 2017.

[7] H. Glauert. The effect of compressibility on the lift of an aerofoil. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 118(779):113–119, 1928.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] N. Gourdain and K. Dewandel. Quantification of the numerical solution sensitivity to unknown parameters: Application to compressor and turbine flows. volume 8, 06 2012.

[10] R. Iriondo. Differences between AI and machine learning, and why it matters. `https://medium.com/datadriveninvestor/1255b182fc6`, Oct 2018.

[11] A. N. Kolmogorov, V. Levin, J. C. R. Hunt, O. M. Phillips, and D. Williams. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 434(1890):9–13, 1991.

[12] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen. The lattice boltzmann method. graduate texts in physics, 2017.

[13] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–, London, UK, UK, 1999. Springer-Verlag.

[14] L. L. Levy Jr. Experimental and computational steady and unsteady transonic flows about a thick airfoil. *AIAA Journal*, 16(6):564–572, 1978.

[15] R. Manceau. *Codes de calcul industriels pour la simulation des écoulements turbulent.* ENSMA, ONERA, 2019.

[16] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015.

[17] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[18] P. Moin and K. Mahesh. Direct numerical simulation : A tool in turbulence research. 1998.

[19] M. A. M. P. H. Cook and M. C. P. Firmin. *Aerofoil Rae 2822: Pressure Distributions, and Boundary Layer and Wake Measurements.* AGARD Report ar 138, 1979.

[20] D. A. Perumal and A. K. Dass. A review on the development of lattice boltzmann computation of macro fluid flows and heat transfer. *Alexandria Engineering Journal*, 54(4):955 – 971, 2015.

[21] D. A. Perumal and A. K. Dass. A review on the development of lattice boltzmann computation of macro fluid flows and heat transfer. *Alexandria Engineering Journal*, 54(4):955 – 971, 2015.

[22] D. Pham and P. Pham. Artificial intelligence in engineering. *International Journal of Machine Tools and Manufacture*, 39(6):937 – 949, 1999.

[23] S. B. Pope. *Turbulent Flows.* Cambridge University Press, 2000.

[24] S. Prasad Kannojia and G. Jaiswal. Effects of varying resolution on performance of cnn based image classification an experimental study. *International Journal of Computer Sciences and Engineering*, 6:451–456, 09 2018.

[25] J. Ruiz and Á. Andrés. *AERODINAMICA BASICA-2 ED.* Serie de ingeniería y tecnología aeroespacial. Garceta, 2011.

[26] M. Ruzicka. *Electrorheological fluids: modeling and mathematical theory.* Springer Science & Business Media, 2000.

[27] R. k. Sampath Kumar, R. Richardson, J. Gustavsson, L. Cattafesta, R. Kumar, Z. Liu, and G. Zha. Characterization of rae 2822 transonic airfoil in fsu polysonic wind tunnel facility. 01 2018.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[29] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[30] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA*, 439, 01 1992.

[31] S. Succi. Lattice boltzmann 2038. *EPL (Europhysics Letters)*, 109, 03 2015.

[32] S. Succi, R. Benzi, and F. Higuera. The lattice boltzmann equation: A new tool for computational fluid-dynamics. *Physica D: Nonlinear Phenomena*, 47(1):219 – 230, 1991.

[33] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3424–3433. JMLR. org, 2017.

[34] I. Viola, V. Chapin, N. Speranza, and M. Biancolini. Optimal airfoil's shapes by high fidelity cfd. *Aircraft Engineering and Aerospace Technology*, 90(6):1000–1011, 2018.

[35] R. Vos and S. Farokhi. *Transonic Similarity Laws*, pages 81–144. Springer Netherlands, Dordrecht, 2015.

[36] L. Wang, X. Zhang, W. Zhu, K. Xu, W. Wu, X. Chu, and w. Zhang. Accurate computation of airfoil flow based on the lattice boltzmann method. *Applied Sciences*, 9:2000, 05 2019.

[37] C. M. Xisto, J. C. Páscoa, P. J. Oliveira, and D. A. Nicolini. A hybrid pressure–density-based algorithm for the euler equations at all mach number regimes. *International Journal for Numerical Methods in Fluids*, 70(8):961–976, 2012.

[38] J. Zhu, L. Liu, T. Liu, Y. Shi, W. Su, and J.-Z. Wu. Lift and drag in two-dimensional steady viscous and compressible flow: I. far-field formulae analysis and numerical confirmation. 06 2015.