POLITECNICO DI TORINO

ENGINEERING FACULTY

Master’s Degree in Aerospace Engineering

Master’s Degree Thesis

A deep learning approach to predicting unsteady airfoil performance from tiny datasets

Supervisor:

Prof. Stefano Berrone

Candidate:

Davide Esposto

Student Number 280241

Co-Supervisor:

Prof. Gaetano Iuso

Academic Year 2023–2024

Abstract

The interaction between fluid and wing surface for the generation of aerodynamic forces is a complex nonlinear phenomenon with multiple variables. Obtaining the aerodynamic performance of a body is, therefore, a complex, time-consuming and costly endeavour. Wind tunnel testing, especially for high-speed flows, requires special facilities and custom-built, high-precision body models. On the other hand, high-precision computational fluid dynamics simulations are expensive, and the possibility of life-like Direct Numerical Simulation for flows of engineering interest remains impractical for the foreseeable future. Especially during the preliminary design phase, exploring the effect of freestream conditions on the aerodynamic performance of a profile requires multiple trials with high time and labour input. In recent years, deep learning has undergone rapid advances in various applications, from computer vision to natural language processing to time series prediction. The inherent ability to extract hierarchical complex dependencies between variables makes neural networks particularly suitable for predicting an airfoil aerodynamic performance. Therefore, in this work, we explore the possibility of predicting the principal harmonics frequency and amplitude of the aerodynamic performance of an airfoil in a high-compressible transonic freestream from near-field data and free streams parameters obtained from numerical simulations. Neural network architectures based on convolutional neural networks are proposed with the goal of learning the data spatio-temporal correlations through the operation of convolution. Compared to similar works, the architectures of the regression models evaluated here are inspired by well-known architectures deployed in the field of image classification and computer vision. The hypothesis is then tested that well-tested computer vision-oriented architectures, due to their feature-extraction capabilities, can be applied to data with different characteristics.

List of Figures

List of Tables

List of Acronyms

AF	Activation Function
ANN	Artificial Neural Network
AoA	Angle of Attack
BC	Boundary Condition
BL	Boundary Layer
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DES	Detached Eddy Simulation
DNS	Direct Numerical Simulation
FFNN	FeedForward Neural Network
FVM	Finite Volume Method
GELU	Gaussian Error Linear Unit
HPC	High Performance Computing
HPO	Hyper-Parameter Optimization
IC	Initial Condition
LES	Large Eddy Simulation
MSE	Mean Square Error
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
RANS	Reynolds Averaged Navier Stokes
ReLU	Rectified Linear Unit
RMSRE	Root Mean Squared Relative Error
RNN	Recurrent Neural Network

List of Symbols

Symbol	Description	Unit
x	Position	m
u	Velocity	$\mathrm{m\,s^{-1}}$
a	Acceleration	$\mathrm{m\,s^{-2}}$
t	Time	s
μ	Coefficient of dynamic viscosity	Pa s
λ	Second coefficient of dynamic viscosity	Pa s
ν	Coefficient of kinematic viscosity	$\mathrm{m^2\,s^{-1}}$
τ	Fluid stress	Pa
ρ	Density	$\mathrm{kg\,m^{-3}}$
p	Pressure	Pa
η	Kolmogorov length scale	m
g	Earth gravitational acceleration	$\mathrm{m\,s^{-2}}$
T	Temperature	K
e	Internal energy	J
E	Total energy	J
$\dot{q}$	Thermal flux	$\mathrm{W\,m^{-2}}$
Q_v	Volumetric source of energy	$\mathrm{W\,m^{-3}}$
κ	Conductivity	$\mathrm{W\,m^{-1}\,K^{-1}}$
c	Speed of sound	$\mathrm{m\,s^{-1}}$
f	Frequency	Hz
$R_{specific}$	Specific gas constant	$\mathrm{J\,kg^{-1}\,K^{-1}}$
c_p	Specific heat capacity at constant pressure	$\mathrm{J\,kg^{-1}\,K^{-1}}$
c_v	Specific heat capacity at constant volume	$\mathrm{J\,kg^{-1}\,K^{-1}}$
γ	Heat capacity ratio	
δ	Kronecker's delta	

Symbol	Description	Unit
Re	Reynolds Number	
M	Mach Number	
Fr	Froude Number	
Pr	Prandtl Number	
St	Strouhal Number	
$C_{\mathbf{L}}$	Coefficient of lift	
$C_{\mathbf{D}}$	Coefficient of drag	
$C_{\mathbf{M}}$	Coefficient of pitching moment	
$C_{\mathbf{p}}$	Pressure coefficient	
$\langle\cdot\rangle$	Ensamble average	
$\tilde{\cdot}$	Favre average	
$\frac{\partial}{\partial}$	Partial derivative	
λ_r	Regularization rate	
w	Neural network connection weight	

Contents

Chapter

Introduction 1

Deep learning has received considerable attention in recent years due to a significant increase in model performance, particularly in fields such as computer vision and natural language processing, time series prediction, art generation, etc. The ability of neural networks to extract complex hierarchical relationships between training data[1,2] and to act as universal approximators of functions[3,4] have made their use attractive in both industry and research to tackle a wide variety of complex problems. Not surprisingly, the capabilities of neural networks have attracted interest from the fluid dynamics research community[5]. Different classes of ANNs have been used for different purposes, such as predicting the aerodynamic performance of airfoils[6–12], determining the free parameters of turbulence models used to solve the RANS equations[13,14], calculation of pressure and velocity distributions in an airfoil near-field[15,16], improving the accuracy of RANS simulations[17], determining the physical properties of a medium from flow data[15,18], direct numerical resolution of the Navier-Stokes equations and other Partial Differential Equations (PDE)[19], as well as more direct applications such as predicting fluid behaviour in a range of different engineering problems[18,20].

At the moment, the use of high-accuracy CFD simulations and wind tunnel testing remains necessary in the industrial environment during the design process, despite the existence of reduced order models, built from the results of high-fidelity simulations, that reduce the complexity of mathematical models of complex dynamic systems to allow a reduction in the computational cost of simulations while maintaining acceptable accuracy. Therefore, investigating the effects of upstream freestream conditions during the preliminary design phases remains an indispensable, costly process regarding economic resources, labour, and time. Deep learning and, more generally, all data-driven approaches (e.g. random forests, etc.), with their capability to learn complex relationships within the training data, are ideal candidates for producing reduced-order models of complex and multivariable systems, such as the fluid-surface interaction for the generation of aerodynamic forces. Accordingly, the ability of neural networks to predict the aerodynamic performance of airfoils has been repeatedly investigated in recent years, using different classes of ANN and under different conditions.

In this work, the feasibility of predicting the aerodynamic performance of an airfoil immersed in a free stream in the high-compressible and transonic regimes is therefore explored, using as input the scalar parameters characterizing the free stream (i.e. AoA, Re, and M) and near-field flow data collected on an envelope in proximity to the airfoil. The aim is to simulate a hypothetical experimental wind tunnel setup consisting of a rake positioned in the wake and near-field of the airfoil. As this work is a proof of concept (at the time of writing, the author is not aware of any similar work), the density of data collection points (see section 3.6) is significantly higher than would be possible in a real-world setting without inducing significant flow obstruction: However, it should also be noted that the sampling rate of the data collection (~3 kHz; see section section 3.6) is well below the sampling rates of transducers commonly used in wind tunnel testing.

For this purpose, convolutional architectures derived from well-known image classification models are proposed and adapted to the regression problem under investigation. As discussed in section 2.2, the drag and lift produced by a body immersed in a uniform flow are a function of the freestream characteristics and the geometry of the body (i.e. the shape and angle of attack) and, as such, it is common in the literature to use freestream parameters as additional scalar inputs to the network; the proposed architectures are, therefore, adapted accordingly.

This work is structured as follows: the next chapter provides a basic introduction to the problem. Then, the methodology followed to obtain the simulation data is explained. Next, the architecture of the neural networks and the choices of the hyperparameters made are explained. The training of the networks' methodology and the network optimization process are then described. Then, the results are included, and finally, a concluding section closes the work.

1.1 Challenges of assembling a large dataset

As previously mentioned, CFD simulations remain expensive in terms of time and computational resources, while wind tunnel testing requires dedicated facilities and custom-built high-accuracy models. Consequently, analysing multiple cases and generating a large dataset requires a non-negligible use of resources in both industrial and research applications. The difficulty of assembling large datasets is encountered in several fields, such as engineering[21,22] or medicine[23–25], where the cost, time, or rarity of the phenomenon prevents obtaining a large number of samples[26]. The present work is focused on creating simulations of real-world situations while dealing with constraints such as limited resources and time. The main objective is to investigate the difficulties of working with small datasets and to develop models that can accurately predict outcomes even with limited training samples. State-of-the-art deep-learning models use datasets composed of millions, if not billions (and arguably, the increase in state-of-the-art model performance over the past years can partially be attributed to the ever-increasing size of training dataset size[26]) of samples, as opposed to the low-hundred samples used in this work. However, it should be noted that the small size of the datasets refers to the number of samples and not to the storage size of the data, as each dataset obtained is approximately 50GB in size.

The decision to use a low number of samples is justified because a hypothetical future use of deep learning for the prediction and design of wings and airfoils would need to be justified by reducing the number of simulations or wind tunnel tests required. Thus, deep learning models that require a large number of training samples would not offer a competitive advantage over current high-fidelity CFD simulations.

Issues arising from a small number of training samples, such as overfitting and the stability of the training process was, addressed by ad hoc methods, such as the use of dropout layers to reduce the risk of overfitting[27] or multiple iterations of the model training process to account for the inherent aleatory nature of the neural network weight initialization process[23]. See Chapter 4 and section 4.12 for more information.

Chapter

Background 2

Below, the theoretical foundations considered essential to the work are presented and briefly explained. Since the proposed work involves both fluid dynamics and neural networks, a summary of the basic concepts of both fields is given. The aim is to make the current work understandable to those unfamiliar with the two fields involved. This will ensure that the reader has a solid understanding of the relevant concepts and can easily follow the work. Later sections in subsequent chapters will provide a deeper theoretical framework where necessary.

2.1 Navier-Stokes equations for a compressible, Newtonian, Fourier fluid

The Navier-Stokes equation is a nonlinear PDE system that governs the motion of viscous, conducting fluids. They express the conservation of fundamental physical properties such as mass, momentum, and energy. The complete derivation of the equations is not shown here as it is considered outside the scope of this work. The equation of conservation of mass, also known as the continuity equation, is shown in eq. (2.1.a); the equation of conservation of momentum, which is a vectorial equation, is shown in eq. (2.1.b); the equation of conservation of energy (in one form of the many reported in the scientific literature) is shown in eq. (2.1.c). It should be noted that the equations reported are valid for a viscous, compressible fluid affected by gravity. Moreover, the system eq. (2.1) shows the Navier-Stokes equations in their differential form, or *strong formulation* as their analytical solution (if any) *requires* the partial derivatives to be fully defined over the entire domain; thus, the strong formulation of the Navier-Stokes equations does not allow any discontinuity over the flow field.

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial}{\partial x_i}(\rho u_i) = 0 & \text{(2.1.a)} \\ \dfrac{\partial}{\partial t}(\rho u_i) + \dfrac{\partial}{\partial x_j}\left(\rho u_i u_j\right) = -\dfrac{\partial p \delta_i}{\partial x_i} + \dfrac{\partial}{\partial x_j}(\tau_{ij}) + \rho g_i & \text{(2.1.b)} \\ \rho\left(\dfrac{\partial E}{\partial t} + u_i \dfrac{\partial E}{\partial x_i}\right) = -\dfrac{\partial}{\partial x_i}(p u_i) + \dfrac{\partial}{\partial x_j}(\tau_{ij} u_i) + \rho g_i u_i + \dfrac{\partial \dot{q}_i}{\partial x_i} + Q_v & \text{(2.1.c)} \end{cases}$$

The true stress tensor σ_{ij} can be expressed as the sum of hydrostatic pressure (i.e. the pressure acting on a fluid at rest) and the viscous stress tensor:

$$\sigma_{ij} = -p\delta_{ij} + \tau_{ij} \tag{2.2}$$

Where hydrostatic pressure is defined as the average normal force on a fluid particle, the average of the true stress tensor trace.

$$-p = \sigma_{jj}/3 = \frac{1}{3}\left(\sigma_{11} + \sigma_{22} + \sigma_{33}\right) \tag{2.3}$$

If the fluid is classified as Newtonian, namely that the fluid is isotropic, that the strain tensor is a linear function of the strain rate tensor, and the divergence of the stress tensor is equal to zero for a resting flow (as $\Delta \cdot \boldsymbol{\tau} = 0$), then the stress tensor $\boldsymbol{\tau}$ can be expressed as

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \delta_{ij}\lambda\frac{\partial u_k}{\partial x_k} = 2\mu S^*_{ij} + \lambda\nabla \cdot \bar{u} I \tag{2.4}$$

Where μ is the fluid's dynamic viscosity, and λ is the so-called second coefficient of viscosity, related to the bulk viscosity of the fluid and to the stress produced by pure compression/expansion of the fluid. It should also be noted that the pure shear strain rate tensor S^*_{ij}, which is the symmetric part of the velocity gradient minus the velocity divergence, has trace equal to zero and, as such, only represents the fluid strain caused by pure shear. It is defined as:

$$S^*_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad , \quad S^*_{ii} = 0 \tag{2.5}$$

Under the commonly held Stokes' Hypothesis, the bulk viscosity is assumed to be zero, and the two viscosity coefficients are related by the expression in eq. (2.6).

$$\lambda = -\frac{2}{3}\mu \tag{2.6}$$

This is correct for monatomic gasses and is a commonly held hypothesis in Computational Fluid Dynamics (CFD)[30]. It follows that an expansion or compression of the fluid produces no viscous effect, that viscous effect does not contribute to hydrodynamic pressure, and that τ_{ii}=0. Even if it is usually used to simplify the Navier-Stokes equations and it makes it possible to obtain accurate results for most flows, Stokes' Hypothesis holds little to no physical basis and can be quite inaccurate depending on the flow and the fluid[31,32].

If the fluid is heat-conducting and can be described by Fourier's law of conduction, the heat flux is proportional to the temperature gradient[33].

$$\dot{q}_i = -\kappa \frac{\partial T}{\partial x_i} \tag{2.7}$$

It should be noted that the minus sign in eq. (2.7) takes into account thermodynamics second law and ensures that heat flows from a higher temperature region to a lower temperature one. Substituting eq. (2.4), (2.6) and (2.7) in eq. (2.1) the Navier-Stokes equations for a compressible, Newtonian, Fourier fluid are obtained. These are shown in eq. (2.8).

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial}{\partial x_i}(\rho u_i) = 0 & \text{(2.8.a)} \\ \rho\left(\dfrac{\partial u_i}{\partial t} + u_j \dfrac{\partial u_i}{\partial x_j}\right) = -\dfrac{\partial p}{\partial x_i} + \mu \dfrac{\partial}{\partial x_j}\left[\left(\dfrac{\partial u_i}{\partial x_j} + \dfrac{\partial u_j}{\partial x_i}\right) - \dfrac{2}{3}\delta_{ij}\dfrac{\partial u_k}{\partial x_k}\right] + \rho g_i & \text{(2.8.b)} \\ \rho\left(\dfrac{\partial E}{\partial t} + u_i \dfrac{\partial E}{\partial x_i}\right) = -\dfrac{\partial}{\partial x_i}(p u_i) + \mu \dfrac{\partial}{\partial x_j}\left\{\left[\left(\dfrac{\partial u_i}{\partial x_j} + \dfrac{\partial u_j}{\partial x_i}\right) - \dfrac{2}{3}\delta_{ij}\dfrac{\partial u_k}{\partial x_k}\right] u_i\right\} & \text{(2.8.c)} \\ \qquad + \rho g_i u_i + \kappa \dfrac{\partial^2 T}{\partial {x_i}^2} + Q_v & \text{(2.8.d)} \end{cases}$$

Two equations of state, which correlate state variables of the thermodynamic system, are needed to guarantee closure of the system of equations. One of these is the ideal gas law (eq. (2.9.a)), as air, which is a mixture of nitrogen, oxygen, carbon dioxide, argon, and other trace substances, can be treated as an ideal gas with reasonable accuracy[34]. The other equation of state, shown in eq. (2.9.b), is the internal energy per unit of mass for an ideal gas.

$$\begin{cases} p = R_{specific}\rho T & \text{(2.9.a)} \\ e = c_v T & \text{(2.9.b)} \end{cases}$$

As shown in eq. (2.9.b) the specific heat capacity is assumed to be constant. Under this hypothesis, the gas is considered as *calorically perfect.* Such a model can be considered valid for a gas temperature up to 2500 K[35], well within the range of temperatures under the scope in this work. Moreover, the heat capacity ratio experiences a variation of ~0.2% in the temperature range 0-200℃[36], so the assumption of a calorically perfect gas is motivated.

It should be noted that, as evident in eq. (2.8), the coefficient of dynamic viscosity and the fluid conductivity were assumed to be constants in space. However, as highlighted in section 3.4, the simulated flow fields show temperature variations in excess of 100 K due to shocks. Therefore, viscosity *cannot* be modelled as constant. The dynamic viscosity dependence on temperature is expressed by Sutherland's Law[37].

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right)^{3/2} \frac{T_{ref} + S}{T + S} \tag{2.10}$$

From now onwards, the previous assumptions are taken for granted, and the equation system in eq. (2.8) is the one that governs fluid behaviour and motion.

2.1.1 Vorticity and vorticity equation

Vorticity is a central aspect of fluid dynamics and holds particular significance to the mechanics of turbulence, as every turbulent flow is necessarily rotational (its vorticity is non-zero). Intuitively, vorticity is related to the spinning motion of the fluid around an arbitrary point. It is defined as the curl of the velocity field.

$$\omega_k = \varepsilon_{ijk} \partial_i u_j = \varepsilon_{ijk} \frac{\partial u_j}{\partial x_i} \tag{2.11}$$

Where ε_{ijk} is the Levi-Civita symbol, defined in three dimension as:

$$\varepsilon_{ijk} = \begin{cases} +1 \text{ if } (i, j, k) \text{ is } (1, 2, 3), (2, 3, 1), \text{ or } (3, 1, 2) \\ -1 \text{ if } (i, j, k) \text{ is } (3, 2, 1), (1, 3, 2) \text{ or } (2, 1, 3) \\ \;\; 0 \text{ if } i = j, \text{ or } j = k, \text{ or } k = i \end{cases} \tag{2.12}$$

As previously stated, the velocity gradient factorized into a symmetric and an antisymmetric matrix thanks to the process of matrix decomposition. The symmetric factor is the previously cited *rate of strain tensor*, while the antisymmetric factor is the rate of rotation tensor.

$$\Omega_{ij} \equiv \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) = -\frac{1}{2}\varepsilon_{ijk}\omega_k \tag{2.13}$$

Which, as can be easily noticed, neatly correlates the velocity field rotation to the flow vorticity.

Vorticity behaviour and evolution can be studied thanks to the *vorticity equation*. It can be derived by applying the curl operator $\nabla \times (\cdot)$ to the conservation of momentum equation (eq. (2.8.b)). The full derivation is lengthy and, therefore, not shown. The full vorticity equation[38] is shown in eq. (2.14):

$$\frac{D\omega_i}{Dt} = \frac{\partial \omega_i}{\partial t} + u_j\frac{\partial \omega_i}{\partial x_j} = \underbrace{\omega_j\frac{\partial u_i}{\partial x_j}}_{\text{Vortex stretching-tilting term}} - \omega_i\frac{\partial u_j}{\partial x_j} + e_{ijk}\frac{1}{\rho^2}\frac{\partial \rho}{\partial x_j}\frac{\partial p}{\partial x_k} + e_{ijk}\frac{\partial}{\partial x_j}\left(\frac{1}{\rho}\frac{\partial \tau_{km}}{\partial x_m}\right) + e_{ijk}\frac{\partial g_k}{\partial x_j} \tag{2.14}$$

The first term is the vortex stretching-tilting term. It is responsible for the stretching of turbulent structures, the onset of instabilities in the turbulent eddies, and the subsequent breakup into smaller structures. As such, it is a fundamental aspect of the mechanics of the energy cascade, later described in section 3.4.2.

2.2 Dimensionless numbers and similitude theory

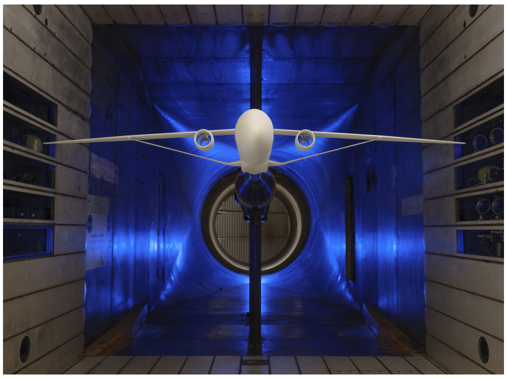


Fig. 2.1: Wind tunnel test of a scale model of a Truss Braced Wing configuration aircraft at NASA Ames Research Center. Source: *NASA*, 2023[39].

Thermodynamic systems, such as the external flow over a rigid body examined in this work, are governed by a large number of parameters. Therefore, designers are tasked with considering the interplay of such parameters when aiming to minimize or maximize any feature (for example, lift, drag, etc.) of the designed body. This amounts to a large amount of testing, both experimental in wind tunnels and numerical with CFD. This would not be ideal and would certainly ramp up the cost and time expenditure of testing and designing. Dimensional analysis, particularly the Buckingham π theorem, helps reduce the number of parameters governing the examined physical system. As such, dimensional analysis is a fundamental tool for designers[40]. The Buckingham theorem states that if a physical system is governed by an equation or a system of equations function on n variables representing physical quantities, then the former equation *can* be rewritten as a function of n-k dimensionless parameters, where k is the number of the fundamental physical unit of measure involved in the system[41]. Strongly correlated to dimensional analysis is the engineering branch known as similitude theory. It states that different physical systems behave the same way as long as the two share the same *shape* and the dimensionless number governing the aforementioned functional relationship of the physical phenomena[42]. Similitude theory is a powerful tool in both testing and design (in the aerospace, automotive, and civil engineering fields, among others), allowing the testing of scaled-down geometry in wind tunnels, in different fluid media, and/or in different ambient conditions[35,40,42]. In eq. (2.15) multiple dimensionless numbers well-known in the fluid mechanics field are shown. It is by no means an exhaustive list, as it only covers those of particular interest for the rest of the work.

$$Re = \frac{uL}{\nu} = \frac{\rho u L}{\mu} \tag{2.15.a}$$

$$Fr = \frac{u}{\sqrt{gL}} \tag{2.15.b}$$

$$M = \frac{u}{c} \tag{2.15.c}$$

$$Pr = \frac{\nu}{\alpha} = \frac{c_p \mu}{k} \tag{2.15.d}$$

$$St = \frac{fL}{U} \tag{2.15.e}$$

Where:

Re = Reynolds number; ratio of fluid inertial and viscous forces

Fr = Froude number; ratio of fluid inertial to gravitational forces

M = Mach number; effect of inertial flow compression; dimensionless velocity

Pr = Prandtl number; ratio of viscous diffusion rate over thermal diffusion rate

St = Strouhal number; ratio of characteristic oscillatory velocity to flow velocity

Dimensional analysis can be applied to the physical system that is the focus of this work: a rigid body hit by a compressible, homogeneous, steady freestream. Such a system is routinely analyzed in both wind tunnels and CFD simulations and, as previously mentioned, this work is attempting to reduce the designer's workload during the design phase and reduce experimental and computational costs. Intuitively, the force F applied by the fluid on the rigid body can be imagined as an unknown function of the parameters shown in eq. (2.16).

$$F = f\,(\text{ geometry}, \ell, V, \rho, T, p, e, \mu, \kappa, T_w, g) \tag{2.16}$$

Applying the Buckingham theorem, the expression can be rewritten as[43]:

$$\frac{F}{\frac{1}{2}\rho\ell^2} = f\left(\text{ geometry}, Re, M, \frac{p/\rho}{c^2}, \frac{c^2}{e}, Pr, \frac{C_pT}{\varepsilon}, Fr, \frac{T_w}{T}\right) \tag{2.17}$$

If the fluid is modelled as an ideal gas, then the following equations hold[35,40]:

$$c^2 = \gamma R_s T, \quad e = c_v T, \quad \gamma = {}^{c_p}/_{c_v} \tag{2.18}$$

then the following parameters can be rewritten as

$$\frac{p/\rho}{c^2} = \frac{1}{\gamma}, \quad \frac{c^2}{e} = \gamma(\gamma - 1), \quad \frac{C_pT}{\varepsilon} = \gamma \tag{2.19}$$

The functional relation is then reduced to: eq. (2.20).

$$\frac{F}{\rho V^2 \ell^2} = f\left(\text{ geometry}, \gamma, Re, M, Pr, Fr, \frac{T_w}{T}\right) \tag{2.20}$$

Under the assumption of a calorically perfect model, γ is a constant. Moreover, in the aerospace field, the effect of gravity acceleration is usually neglected due to a high-velocity field and a high Froude number. The effect of Prandtl number *can* be neglected, and it usually is when the physical system temperature range is narrow enough. In addition, the term $\frac{T_w}{T}$ relative to heat exchange can be neglected as, for every scenario considered, the body surface will be considered adiabatic (see section 3.4.5). The result is:

$$\frac{F}{\rho V^2 \ell^2} = f\,(\text{ geometry}, Re, M) \tag{2.21}$$

Which is the functional relationship between the coefficient of lift and the coefficient of drag. A similar result can be derived for the pitching moment coefficient, but it will be omitted from this section.

The results given by dimensional analysis are notable for developing the neural network architectures, which will be presented in Chapter 4. Mainly, eq. (2.16) refers to a *generic* functional relationship and, as such, offers no insight on the structure of the underlying function and cannot be used to obtain numerical results. However, ANN use training data to learn more and more abstract correlations and hierarchies between the data[2] and therefore ANN are ideally suited to learn the unknown functional relationship shown in eq. (2.16).

2.3 Phenomenology of turbulence

Turbulence is one of the fundamental aspects of fluid dynamics, and most flows in nature and engineering are turbulent (e.g. water flow in a river, the atmospheric boundary layer, the flow on a bird's wing, etc.). Turbulence is such a common phenomenon that it could be argued that laminar flow is an exception. Even so, a formal, rigorous definition of turbulence is lacking[38]. It is instead defined by its empirically observed qualitative and quantitative properties. Turbulent flow presents an inherent non-stationarity, an apparent chaotic behaviour, and eddies (vorticose structures) of different dimensions. The chaotic behaviour of turbulence is such that the velocity cannot be predicted in space or time, and the actual value of the velocity field is random. Obviously, turbulent flow obeys the Navier-Stokes equation, a fully deterministic set of equations[38,44]. The apparent contradiction is resolved by the fact that turbulent flows have an acute sensitivity to the system Boundary Conditions (BC) and Initial Conditions (IC). Even if an experiment or a numerical simulation sets specific BC and IC, the experimental setup will have inevitable perturbations to the nominal ones. Similarly, the numerical simulation BCs and ICs will differ from the nominal ones due to truncation errors, discretization errors, and, ultimately, machine errors. Such sensitivity to BCs and ICs will amount to a different flow realization.

The flow chaotic behaviour is due to its quadratic nonlinearity[38,44], and it can be shown that different nonlinear systems present the same chaotic behaviour. A classic example is the Lorentz System[45], a set of Ordinary Differential Equation (ODE) that, for specific parameters and certain IC exhibits chaotic behaviour. The Lorenz system is shown in eq. (2.22).

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y-x), \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho-z)-y, \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy-\beta z \end{cases} \tag{2.22}$$

Where σ, ρ, β are fixed value parameters. A particular set of solutions to the Lorenz system is known as the Lorenz Attractor. In Fig. 2.2 a visual representation of a sys-

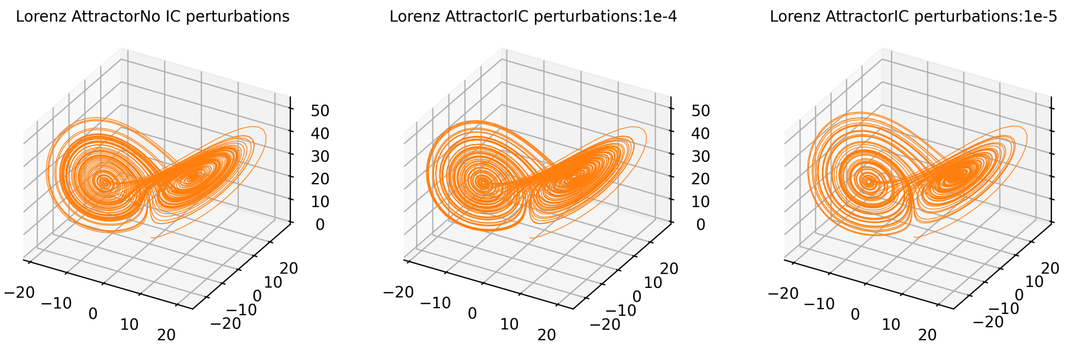


Fig. 2.2: Visualization of a Lorentz attractor for different perturbed Initial Conditions. The parameters value are σ=10, ρ=28, β=8/3. Nominal Initial Condition: (x, y, z)$_{t=0}$=(0, 1, 1.05).

tem's chaotic behaviour is shown by comparing the solution trajectory in the (x, y, z) space at different IC perturbations. As can be seen, while the overall shape of the trajectory is the same, the actual shape is different, even for a very small perturbation of the initial condition. One central aspect of turbulence is that of multi-scalarity (as captured in verses by Richardson in 1922[46][1]) and the process known as the energy cascade. In an essentially inviscid phenomenon, the large eddies (an ill-defined concept that identifies localized coherent turbulent structures[44]) are squeezed and stretched by the mechanism of *vortex stretching* and eventually grow to be unstable, breaking up into smaller turbulent structures. This process is known as *energy cascade* and is responsible for energy flow from the biggest turbulent structures to the smallest ones. The smallest turbulent structures are those where the viscous forces become significant. At this scale, known as the Kolmogorov scale, the turbulent energy is dissipated before eddies break up. The smaller scales are characterized by higher velocities due to the conservation of angular momentum and higher velocity gradients. It can be shown[38] that the range of scales of turbulent flow widens as the Reynolds number rises. In particular:

$$\frac{\eta}{L} = Re^{-\frac{3}{4}} \tag{2.23}$$

Where η is the Kolmogorov scale, and L is the characteristic length of the largest eddies. Such separations of scale make turbulent flow hard to simulate, requiring very fine meshes and short time steps to capture fine flow structures. The numerical method of simulating the entire turbulent flow across all length scales from the direct resolution of the Navier-Stokes equations is known as Direct Numerical Simulation (DNS). DNS resolution of most practical purposes flows is often economically unviable or downright unfeasible with the currently available computational power. Other methods include solving the NS equations through a turbu-

[1] *Big whorls have little whorls,*
Which feed on their velocity;
And little whorls have lesser whorls,
And so on to viscosity
(in the molecular sense).

Lewis F. Richardson (1922)

lence model, a mathematical model to predict the average motion of turbulent flow. The most common turbulence models are Large Eddy Simulation (LES), Detached Eddy Simulation (DES) and RANS. The turbulent model used in this work is a RANS model. The reasoning behind such a choice is expanded upon in section 3.4.2.

2.4 Boundary layer, transition and separation

Fundamental to the dynamics of fluids over solid surfaces is the presence of the *boundary layer*. On a solid body surface, due to the adhesion between the fluid molecules and the body surface, the fluid is at rest relative to the surface. This is the so-called *no-slip condition*, which acts as a Dirichlet BC for the Navier-Stokes equations. Far from the body surfaces, the flow velocity fields U_e will depend on the freestream velocity U_∞. Due to the effect of fluid friction on the fluid in the layers just above and below any point of interest, we anticipate the velocity profile to shift smoothly between 0 on the body surface and U_e far from the surface. Therefore, near the body surface, a region develops: the boundary layer (also known as the momentum or velocity boundary layer). Its defining characteristics are high-velocity gradients in the direction normal to the body surface and the importance of the viscosity terms of the Navier-Stokes equations[47]. The boundary layer thickness can be defined in various ways, but it is commonly considered equal to the distance δ at which the fluid reaches 99% of U_e.

Similar changes will occur in the flow temperature above the wall, ranging from wall temperature T_w on the surface body to the temperature of the external flow at a distance δ_T. This region is called the *thermal boundary layer*. The actual temperature profile is determined by thermal conduction, frictional heating, and kinetic energy dissipation due to viscous effects[40]. The thermal energy transfer between wall and fluid is generally referred to as *aerodynamic heating*[40]. This phenomenon is generally considered negligible in subsonic flows but becomes increasingly important at high speeds, particularly in the supersonic regime[40]. The flow considered in this work is in a compressible transonic regime; therefore, frictional heating is considered an essential factor in thermal energy transport. Moreover, the effect of thermal conduction will be disregarded as the airfoil surface is set to be adiabatic and, as such, no heat is transferred from the wall to the fluid. The effect of the compressible regime on the development of the boundary layer can be summarized by the fact that the rapid changes in density and temperature (especially in the presence of shocks) do not allow density and viscosity to be considered constants even within the limited region of the boundary layer[35,47].

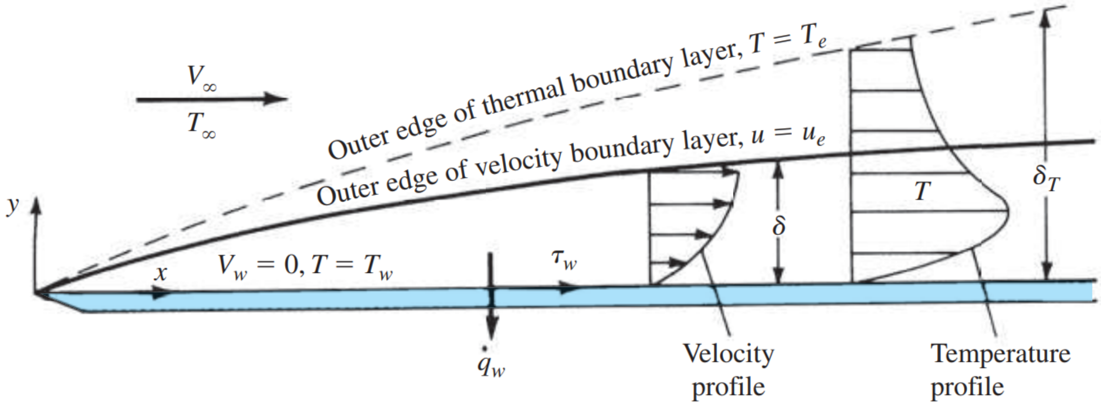


Fig. 2.3: Illustration of the velocity boundary layer and the thermal boundary layer over a smooth flat plate. The wall depicted is non-adiabatic, and the wall temperature is lower than the external flow temperature (that is, $T_w<T_e$).Source: *Anderson*, 2017[40].

In Fig. 2.3, the development and transition to turbulence of the boundary layer over a flat, smooth plate are shown. Notably, every momentum boundary layer initially develops as laminar and eventually transitions, over a region, to fully turbulent flow[40].

Examining laminar or turbulent boundary layer regimes is essential to understanding how boundary layer theory is applied in engineering. This question is more than relevant since there are significant differences between the two scenarios in terms of, for instance, the amounts of skin friction and heat transfer. This variation is caused mainly by the significantly enhanced mixing brought on by the turbulent eddies' unpredictable velocities. The first thing that is crucial to grasp about transition is that it is a process that involves numerous steps that take place throughout a region of the flow rather than a single, abrupt occurrence in the flow. It is therefore of paramount importance to know the boundary layer regime in order to predict its behaviour correctly and consequently calculate the values of the aerodynamic wall forces and the resulting flow evolution; for example, the e^N approach and the hydrodynamic stability theory are two of the developed techniques. Since there is yet to be a comprehensive and thorough theory of the process, predicting when the boundary layer will change from laminar to turbulent is not an easy task. The γ-Re_θ model, which is employed in the current work, is based on the idea of intermittency and models the boundary layer transition by calculating the percentage of time that the flow is turbulent (see section 3.4.3.1). In this method, the intermittency γ of the flow (in this case, the boundary layer) will equal 1 after the turbulent flow transition is complete[47].

2.4.1 Wall variables and the turbulent boundary layer internal structure

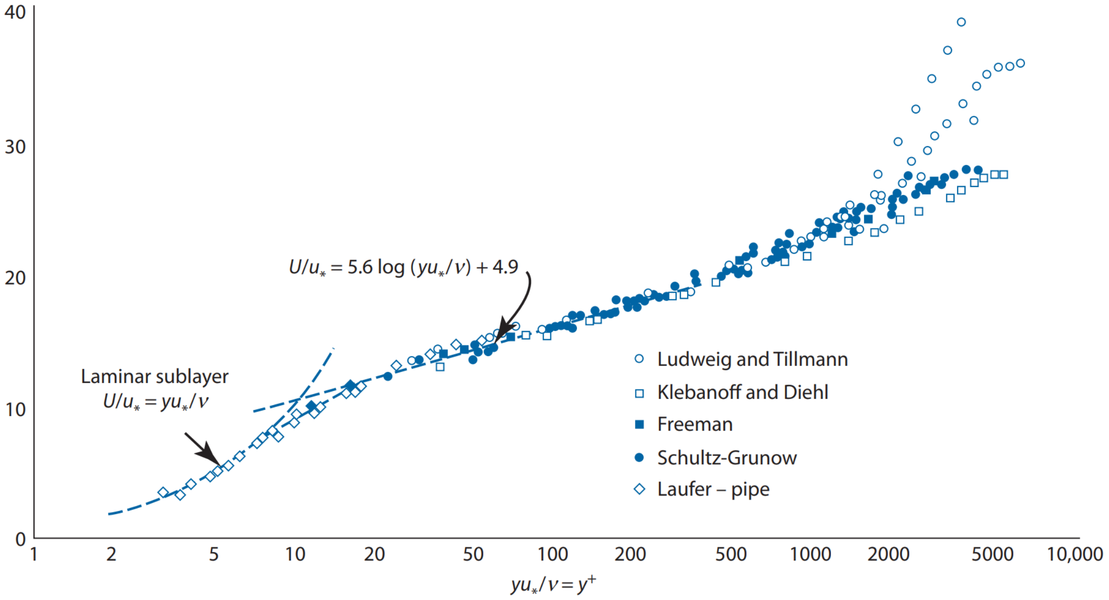


Fig. 2.4: An illustration of the universal law of the wall for a turbulent boundary layer over a flat solid wall . Source: *Rodney D. W. Bowersox*, 2011[47].

For a pure shear wall flow, which can be seen as a simple model for a generic flow over a wall, the momentum balance equation in the mean flow direction can be simplified by neglecting the divergence of the streamlines, as in eq. (2.24).

$$0 = -\frac{\partial\langle p\rangle}{\partial x} + \frac{\partial}{\partial y}\left(\mu\frac{\partial\langle u\rangle}{\partial y} - \rho\,\langle u'v'\rangle\right) \tag{2.24}$$

The shear stress along the boundary layer can be assumed to be constant and equal to the wall shear stress if the boundary layer is assumed to have a limited size. This hypothesis is reasonable in general and true for a zero pressure gradient flow. Therefore, we can generally say that:

$$\mu\frac{\partial\langle u\rangle}{\partial y} - \rho\,\langle u'v'\rangle = \tau_w \tag{2.25}$$

Since the shear stress near the wall is constant, a velocity scale can be defined, called the frictional velocity $u_* = \sqrt{\frac{\tau_w}{\rho}}$, and a length scale can be similarly defined, called the viscous length $l_v = \frac{\nu}{u_*}$. This allows two new flow variables to be defined, called *wall variables*:

$$u^+ = \frac{u}{u_\tau}, \qquad y^+ = \frac{y}{\ell_v} = \frac{yu_*}{\nu} \tag{2.26}$$

We call the viscous sublayer the near-wall region dominated by viscous transport over turbulent fluctuation transport (in other terms, viscous stresses dominate over Reynolds' stresses). This region will be a small fraction of the boundary layer since, globally, the viscous stresses are negligible compared to the convective terms (and thus to the Reynolds' stresses). Neglecting the Reynolds stress term and substituting the wall variables in eq. (2.25), the result is:

$$\langle u^+ \rangle = \langle y^+ \rangle \tag{2.27}$$

Which is valid for $y^+<5$.

On the contrary, far from the wall, the shear stress can be assumed to be the result of Reynolds stresses only. Applying Prandtl's mixing length theory[38], the shear stress away from the wall can be rewritten as in eq. (2.28).

$$\tau_p \simeq -\rho \langle u'v' \rangle \simeq \rho \kappa^2 y^2 \left(\frac{\partial \langle u \rangle}{\partial y} \right)^2 \tag{2.28}$$

Assuming κ has a constant value, integrating eq. (2.28) and substituting the frictional velocity and wall variables, the result is:

$$\left\langle u^+ \right\rangle = \frac{1}{\kappa} \log y^+ + C \tag{2.29}$$

Where κ and C are constants, this is known as the logarithmic law of the wall, which describes the dimensionless velocity profile in the so-called logarithmic layer and is valid for $30<y^+<100$. Eq. (2.27) and (2.29) are shown in Fig. 2.4 and are almost universally valid in the case of a zero pressure and describe turbulent shear flows close to solid boundaries.

Outside the logarithmic layer, equation eq. (2.29) ceases to be valid and must be adjusted through the wake function W, which is flow-dependent[38,47]. In eq. (2.30), the generic form of a corrected logarithmic law is given without expressing the wake function.

$$\frac{U}{u_*} = \frac{1}{\kappa} \ln \left(\frac{y u_*}{\nu} \right) + C + \frac{\Pi}{\kappa} W \left(\frac{y}{\delta} \right), \quad \text{where} \quad W = \left(\frac{y}{\delta} \right) = 2 \sin^2 \left(\frac{\pi}{2} \frac{y}{\delta} \right) \tag{2.30}$$

The law of the wall can be used to numerically solve the turbulent boundary layer[30,48]. However, in the present work, the turbulence model adopted (see section 3.4.2) allows the boundary layer to be solved directly by integrating the equations of motion. Nevertheless, it remains necessary to ensure that the spatial resolution of the mesh is such that the centroid of the first wall-adjacent cell has a distance from the wall of approximately y^+=1 or less[48] (see section 3.4.4).

2.4.2 Separation and reattachment

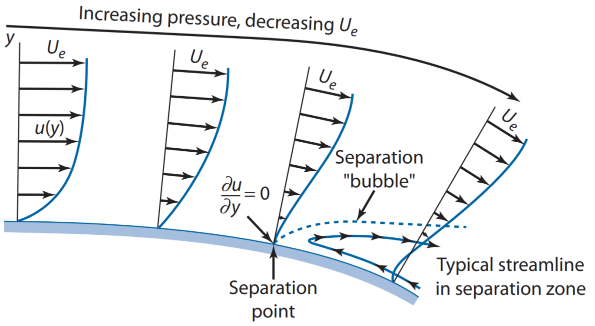

Increasing pressure, decreasing U_e
y
U_e
U_e
U_e
U_e
$u(y)$
$\frac{\partial u}{\partial y} = 0$
Separation "bubble"
Typical streamline in separation zone
Separation point

Fig. 2.5: Pressure velocity profile of a laminar boundary layer under an adverse pressure gradient ($\frac{\partial p}{\partial x} > 0$) and consequent flow reversal and BL separation, Source: *Rodney D. W. Bowersox*, 2011[47].

Flow separation refers to the detachment of the boundary layer from the surface of a body. Separation is associated with an adverse pressure gradient at the body's surface (thus, for a positive AoA, generally at the upper surface of an airfoil). As shown in Fig. 2.5 the presence of a positive pressure gradient leads to a reduction in the velocity of the fluid particles at the wall. Eventually, these particles arrive at the separation point with zero velocity with respect to the wall and reverse their direction of motion: the fluid particles upstream of the separation point collide with the stationary flow, which acts as an effective obstacle that is the reverse flow, and the flow separates.

Boundary layer separation is associated, in airfoils, with a sharp reduction in lift and an increase in drag. This is due to the presence of vortices and recirculation bubbles in the separated flow zone, which cause a drastic change in the pressure distribution. Furthermore, the assumptions underlying the boundary layer theory cease to be valid, and the entire recirculation zone is strongly influenced by viscous phenomena[47]. If the flow reattaches downstream of the separation point, this is referred to as boundary layer reattachment. It leads to the formation of a *separation bubble* (or recirculation bubble), characterized by approximately constant pressure.

In the case of high-velocity flows, the presence of supersonic shocks, which act as discontinuities in the pressure field, can cause flow separation. This is referred to as *shock-induced separation*. In the cases simulated in this work, boundary layer separation is expected at high AoA and at high Mach numbers due to the presence of shocks. The resulting drag increase and lift decrease due to a change in the pressure distribution[40] (also referred to as pressure drag) is also expected.

It should be noted that turbulent flows are less affected by adverse pressure gradients. That is, a turbulent boundary layer is subject to further downstream separation than a laminar boundary layer[47]. Consequently, the boundary state transition point and its relative position to the adverse gradient zone directly influence boundary state separation. For this reason, it was considered necessary to accurately predict the transition point of the boundary layer from laminar to turbulent using the γ-Re$_\theta$ model (see section 3.4.3.1).

2.5 Artificial Neural Networks and Deep Learning

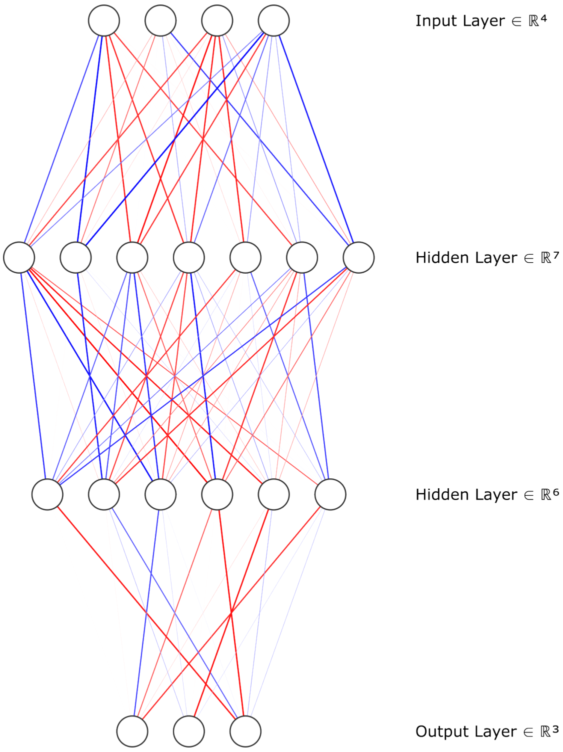


Fig. 2.6: Visualization of a randomly generated multilayer FFNN. Negative weights are shown as blue, and positive weights are shown as red. Line width visually represents the weight's magnitude.

ANNs are a machine learning model loosely based on the structure and function of animal brains[1,49]. The idea behind neural network engineering is that the brain is capable of abstract and intelligent reasoning. Emulation and reverse engineering of the brain's underlying functioning is, therefore, one of the possible approaches to emulating human cognition. This is closely related to the common use of neural networks in problem-solving: tackling problems intuitive for biological agents, such as vision or natural language processing, but have been considered problematic for computers to solve for decades. This is because tasks considered ”natural” or trivial by people actually require extensive knowledge that is considered intuitive but complex for a machine to formalize.

The approach to engineering systems capable of extracting patterns and hierarchical relationships from raw data, often used for solving problems ill-fitted to a traditional hard-coded approach to software development, allows systems to acquire their own knowledge and is known as *machine learning*. This approach to formalizing human cognition through connected networks was hypothesized in the 1930s and dates to the 1950s and is known as Connectionism[49]. The research community's interest in Connectionism has had several waves over the years, starting with the Perceptron by Mcculloch and Pitts (1943)[50]. The second wave began in the 1980s with the conception of the Multi Layer Perceptron[51] (also known as FeedForward Neural Network (FFNN), the most straightforward modern neural network architecture, the operation of which will be expanded upon later) and training via backpropagation. However, interest died down due to the unrealistic expectations set up at the time. The third wave consists of the current popularization of neural networks in Deep Learning, popularized by the introduction of newer, less computationally expensive training algorithm[49], generalized availability of computing power and the opportunity to obtain and manage massive data sets. Specifically, *Deep Learning* refers to using complex models to represent systems and tasks through multiple levels of abstraction, allowing deep learning to uncover complex structures in massive data sets.

A complete and exhaustive history of neural networks and deep learning is not the purpose of this work. For a more in-depth discussion, we refer to other works[52,53].

The elementary unit of a neural network is known as neuron (or *Perceptron*), in analogy to the biological neurons part of the animal brain. The single neuron, as shown in eq. (2.31), obtains an input or inputs, applies an activation function (see section 2.5.3), sums a constant value, called *bias*, and returns an output, as shown in Fig. 2.7. The neuron's single input is multiplied by the connection weight $\mathrm{w_i}$, which can be viewed in analogy to biological brains as the strength of the synaptic connection.

$$z = \sum_{i=1}^{N} a_i w_i \tag{2.31}$$

$$a_{out} = F(z) + b \tag{2.32}$$

Again, in analogy to animal brains, a single neuron does not have high computational capabilities and is not capable of complex problems. An increase in complexity is achieved by increasing the number of neurons by arranging them in a layer. Here, multiple neurons are fed the inputs, and the output of single neurons is again calculated as in eq. (2.31). Each neuron will be connected to the inputs by its own connections, each with its weight, leading to an increase in the free parameters of the model and its complexity. A further increase in model complexity can be achieved by stacking multiple layers of neurons. In this case, the input of the neurons of each specific layer corresponds to the output of the previous layer. The

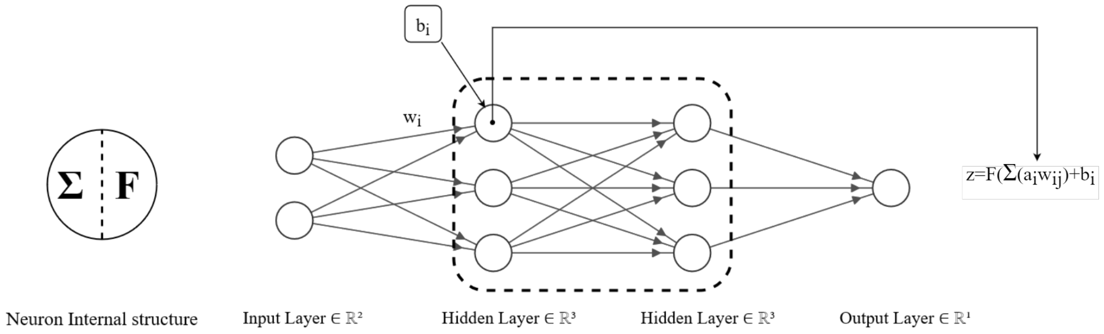


Fig. 2.7: Simple feedforward ANN architecture. The internal structure of the nodes is highlighted and a bias node is shown. Note that bias nodes are usually not shown.

obtained network is called FeedForward Neural Network (FFNN), whose network structure is shown Fig. 2.6. By updating the weights of the connections between neurons, the network is able to learn. This occurs during the training process, in which by analyzing the training samples, comprising both inputs and outputs, the network learns to associate an input with a specific output. Once the training process is over, the network can predict the output when provided with a novel, never-seen-before input. During the training of a neural network, its weights and thresholds start as random values. The training data enters the input layer and travels through subsequent layers. The data then reaches the output layer. Throughout the training process, the network adjusts its weights until data with matching labels produces similar outputs.

A neural network can thus be conceptualized as a parametric function with domain R^{n} (where n is the number of neurons in the input layer) and codomain R^{m} (where m is the number of neurons in the output layer)

$$f{:}R^n \longrightarrow R^m \tag{2.33}$$

where the weights and biases of the layers are free parameters. As a matter of fact, neural networks are considered universal approximators of functions, and ideally, a single-layer network of infinite width can exactly approximate any given function[3,4]. Given a task to be learned, which can be represented as a generic function *f*, the learning process is to find the number of free parameters of the function (commonly called *learnable parameters*) such that the error, as defined by a loss function

$$|f(x) - LS(x)| \tag{2.34}$$

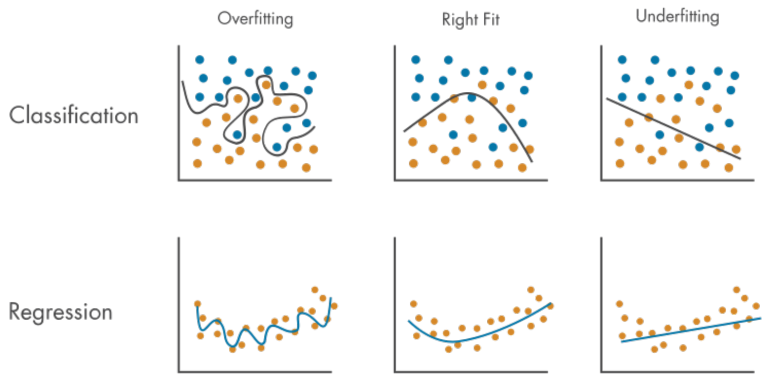


Fig. 2.8: Illustration of the underfitting and overfitting problem for both regression and classification problems. Source: *MathWorks, Inc.*, 2023[55].

is minimized for each x in the domain (where the x in the domain corresponds to the values of the input layer). The training process can then be reframed as a problem of optimizing the error defined by the loss function with respect to the input-output pairs provided in the training examples. This is done using optimization algorithms called *optimizers*, which compute the derivative of the loss function through error backpropagation algorithms, generally based on gradient because the conformation of the NNs is optimal for computing the derivatives through automatic differentiation[54]. Usually, gradient of the last layer is used to calculate the value of the gradient in the previous layers in a backward fashion until the input layer is reached. Having obtained the derivative of the loss function with respect to the dimensional space of the weights, the weights are then updated in such a way as to reduce the value of the loss function, as shown in eq. (2.35).

$$w_j = w_j - \mathrm{lr} \cdot \partial \frac{LS}{\partial w_j} \tag{2.35}$$

Where lr stands for the learning rate, and LS stands for the value of the loss function.

2.5.1 Underfitting and overfitting

The main challenge in training neural networks comes from the fact that the error minimized during training is calculated with respect to the training set (i.e., the input-output pairs used during training). However, the goal during training is to obtain a model from the generalized predictive capabilities. That is to obtain a model capable of solving the considered task for the widest possible range of inputs, especially ”new” data never seen before during the training. The model's predictive capability is measured by calculating the loss function on an unused test set to train the model. When a neural network fails to capture the underlying patterns or complexities in the training data, it is called *underfitting*. Essentially, the model is too simple to capture the nuances within the data. As a result, it performs poorly not only on the training data but also on new, unseen test data). It often happens when the model is too basic and lacks capacity (which can be measured by the number of trainable parameters) or when the model is undertrained and the training process is stopped prematurely.

On the other hand, *overfitting* happens when the model learns not only the underlying patterns but also noise or irrelevant details in the training data. The model becomes too specific to the training data and does not generalize well to new, unseen data. Essentially, it memorizes the training data instead of learning the actual relationships between inputs and outputs. Overfitting occurs when the neural network becomes excessively complex (too much capacity and learnable parameters) or overtrained, capturing details specific to the training set but not applied to new data.

An intuitive representation of both underfitting and overfitting is shown in Fig. 2.8. In both cases, the goal is for the model to learn the essential patterns in the data without being too simplistic (*underfitting*) or overly accustomed to the training dataset (*overfitting*), aiming for generalized predictive performance. Techniques such as regularization, dropout, using more data, or adjusting the model's complexity are often employed to mitigate these issues.

2.5.2 Regularization

As previously mentioned, regularization is used to avoid overfitting the model to the training data, thus increasing the network's generalization capabilities. Two of the most commonly employed techniques, Dropout and L_2 *regularization*, are used in this work and are discussed below.

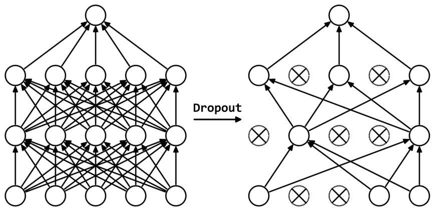


Fig. 2.9: Visualization of Dropout regularization. Source: *Srivastava et al.*, 2014[56].

Dropout regularization works by randomly dropping certain neurons and their weights during training, which improves network accuracy and reduces the risk of overfitting by introducing noise during the training process and by reducing the amount of co-adaptations between multiple neurons inside the neural networks. Neuron dropout was shown to reduce the risk of overfitting in small datasets[57] and even improve network performance for small datasets[26], making it suitable to use in this work. The method is implemented in TensorFlow as a layer[58], which randomly sets a percentage, known as the *dropout rate* of the input neurons to zero. The network obtained has fewer free parameters (due to weight dropping) and has higher *sparsity*.

L_2 regularization lowers the complexity of the model and the risk of overfitting by reducing the absolute value of the model weights. This is achieved by adding an extra term to the loss function. The extra term is:

$$\lambda_r \sum_{j=1}^{p} \beta_j^2 \tag{2.36}$$

where β is one neuron weight and λ_r is the *regularization rate*.

2.5.3 Activation Functions

As previously mentioned, the inputs of each neuron are passed to an Activation Function (AF). Over the years, numerous activation functions have been proposed and applied to neural networks. Some of the existing activation functions are depicted in the figure. The choice of the AF used is critical to the properties of the neural network and the field of application of the model. Consequently, some AFs are mainly used for certain categories of neural networks, i.e., ReLU being the first choice for CNNs, TanH being the first choice for Recurrent Neural Networks (RNN), Linear being used for the output layer of regressive models, or Softmax being a common choice for classification models.

Three different AF are used in this work: ReLU, GELU, and SWISH. The rectified linear unit activation function, commonly known as ReLU, was introduced by Fukushima (1975)[59] and has since been the most used activation function, especially in convolutional networks[60] Despite its widespread use and computational inexpensiveness, ReLU suffers from the ”Dying ReLU Problem”. If, during training, the neuron with the ReLU activation function has a negative input, the neuron will have an output of zero and will, therefore, be deactivated. Furthermore, since the

ReLU function has a constant gradient of zero for input values less than zero, the neuron cannot be reactivated during backpropagation[61,62].

In contrast, GELU and Swish are designed to have a defined gradient for input values less than zero to reduce the problem of dying neurons while maintaining the unboundedness of the output to avoid saturation problems. Both activation functions are used in state-of-the-art deep-learning models. For example, Swish[63] activation is used in the EfficientNet[64] family of models, known to be particularly deep and accurate. GELU[65], on the other hand, is used in the field of Natural Language Processing and is utilized by models in the Generative Pre-trained Transformers (GPT)[66] family. Both GELU and Swish have been shown to outperform ReLU in various image classification tasks[67], and Swish has been shown to be suitable for particularly deep networks[60]. It must be specified how the AF used for the hidden layers must be a nonlinear function; otherwise, the neural network will only be able to learn linear models.

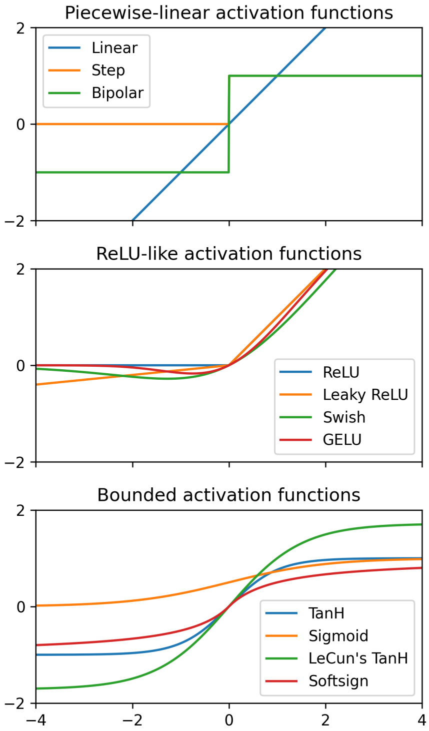


Fig. 2.10: Illustration of some of the most common activation functions used in regressive neural networks.

2.5.4 CNN

Convolutional Neural Networks are arguably the most popular class of neural networks: they have been mainly applied to the fields of image recognition, image segmentation, image classification, and natural language processing. CNNs are neural networks that employ the convolutional operation (as defined by the machine learning community) in the network architecture. It should be noted that the definition of the convolution operation as used in machine learning does not correspond to the one used in mathematics. As can be seen in eq. (2.37), the definition used in machine learning actually corresponds to the mathematical definition of cross-correlation[49,58] which differs from the definition of convolution commonly used in engineering and pure mathematics (eq. (2.38)).

$$s(t) = \int x(a)w(t-a)da \tag{2.37}$$

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.38}$$

In the machine learning community, the input of a convolutional operation (actually a cross-correlation) is applied to N-dimensional matrices (usually referred to as *tensors* in contrast with the actual definition of these mathematical objects) where the first tensor is referred to as input and is usually of bigger size and dimension, while the second matrix is referred as *kernel.* The operation of cross-correlation, which will be from now thereon be referred to as convolution, can be visualized as a dot product between an N-dimensional input matrix and an N-dimensional sliding filter (the *kernel*) with a smaller size than the input matrix as shown in Fig. 2.11. The output of this convolution operation is commonly referred to as *feature map.*

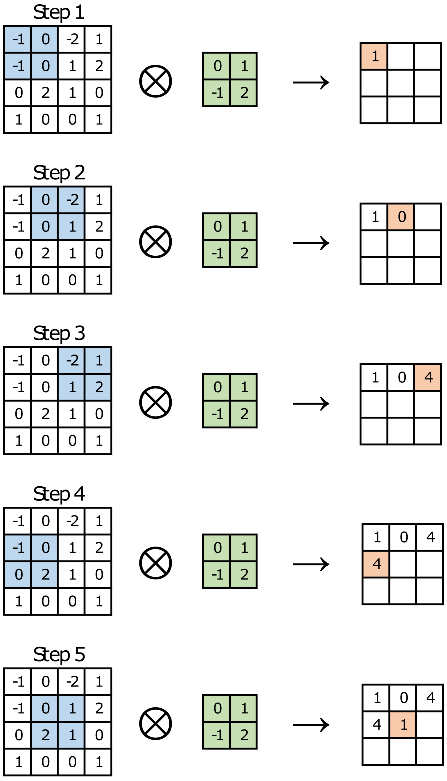


Fig. 2.11: Example of the convolution operation on a matrix of dimension 4×4, 2×2 filter, and unitary stride.

Note that The filters of neural networks can be conceptualized as the neurons of a feedforward neural network; however, unlike the neurons of a feedforward network, which are connected to each neuron of the next layer, the filters of a convolutional layer have a limited receptive field, due to filter size being commonly smaller compared to the layer's input matrix. Consequently, this leads to a reduction in the number of learnable parameters and a reduction in the training time and computational cost of the model, as the convolution operation reduces the dimensionality of the data. Furthermore, the same filter/kernel is used across the input data at multiple locations. This means that the weights of a filter remain constant and are shared across the entire input space. Consequently, the filter learns to detect a particular pattern or feature (like edges, textures, etc.) irrespective of its location in the input image. This concept is known as *weights sharing*, leading to a further reduction in the number of learnable parameters and increased model effectiveness.

The typical configuration of a Convolutional Neural Network (CNN), illustrated in Fig. 2.12, follows a specific pattern. The input layer receives raw data, usually im-

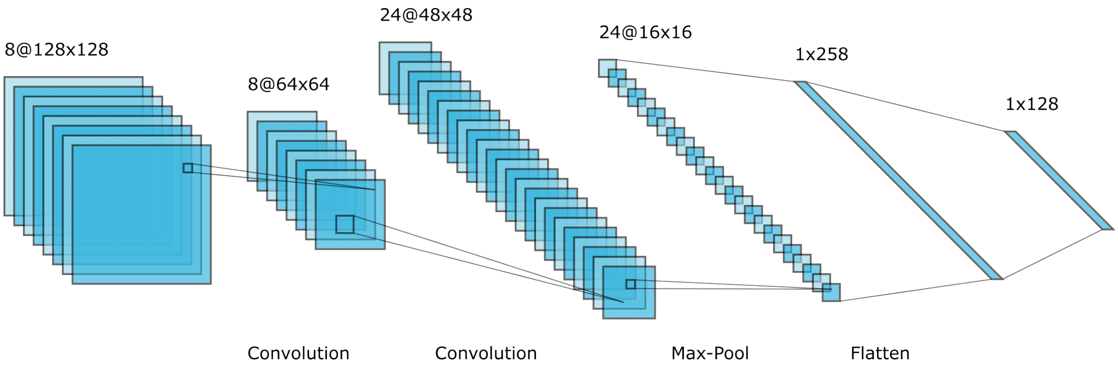


Fig. 2.12: Illustration of a basic CNN architecture, consisting of convolutional layers, max-pooling layers and fully connected (Dense) layers.

ages represented as matrices or tensors. Subsequent layers comprise convolutional layers, wherein adaptable filters or kernels convolve across the input, producing feature maps that capture diverse patterns. Activation functions are then applied to introduce non-linearity, aiding in understanding complex data relationships. Following this, pooling layers decrease feature map dimensions, preserving crucial information while minimizing computational load and the risks of overfitting.

Additionally, fully connected layers are employed for tasks like classification or regression, establishing connections between neurons across layers to comprehend relationships among high-level features. A flattening step reshapes the extracted features into a suitable vector format to prepare for the dense layers. Finally, the output layer provides the ultimate network prediction, configured based on the task, utilizing softmax for multi-class classification or linear activation for regression. This sequential arrangement characterizes the CNN basic architecture.

The main advantage over other classes is the automatic extraction of features from the data during the training process. The output of the convolution operation between the input matrix and learned weights consists of a feature map: multiple convolutional layers produce feature maps representing increasingly complex and abstract hierarchical relationships between the input data[1,68]. Therefore, the operator-based and error-prone feature extraction necessary on pure FFNNs is no longer needed. Furthermore, the mechanism of shared weights due to sliding filters leads to a reduction of model parameters and, thus, a reduction of computational cost and training time[68]. It should be noted that successful CNN architectures, such as the winning models of the ImageNet Large Scale Visual Recognition Challenge[69], generally employ one or more fully connected layers downstream of the convolutional architecture in order to obtain the output. Similarly, this work employs hybrid CNNs architectures inspired by image classification architectures, with fully connected layers for output generation. This choice was made because CNNs allow capturing complex patterns in the training data, which also applies to the spatio-temporal relationships in the training dataset used in this work. Furthermore, given the small dataset used, special attention was paid to the ability of convolutional networks to obtain capable models with a low number of learnable parameters, thanks to the mechanism of sparse connections and weights sharing, reducing the risk of overfitting to the training data. This is expanded upon in Chapter 4.

2.5.4.1 Padding

The convolution operation leads to a reduction in the dimensionality of the input. The padding operation is introduced to keep the tensors' dimensionality constant and ensure the full convolution of the edge pixels. The size of the output of the convolution operation is given in eq. (2.39).

$$L_w \times L_h \quad \Longrightarrow \quad \left[(L_w - K_w + 2 \cdot P_w) / S_w + 1 \right] \times \left[(L_h - K_h + 2 \cdot P_h) / S_h + 1 \right] \tag{2.39}$$

Where $\mathrm{L_w}$, $\mathrm{L_h}$ are the input tensor width and height, $\mathrm{K_w}$, $\mathrm{K_h}$ are the kernel width and height, $\mathrm{P_w}$, $\mathrm{P_h}$ are the padding width and height and $\mathrm{S_w}$, $\mathrm{S_h}$ are the stride width and height. Assuming a unitary stride, the padding required to maintain tensor dimensionality can be easily calculated.

2.5.4.2 Pooling

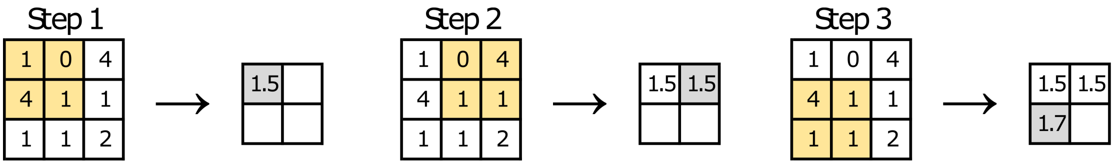


Fig. 2.13: Representation of the average pooling operation on a matrix of size 2x2.

Some of the proposed models use the pooling operation, shown in Fig. 2.13, to reduce the dimensionality of the feature maps produced by the convolution operations. Thus, while the role of the convolution operation is to extract features, the role of the pooling operation is to perform the merging of similar features. Consequently, the pooling operation reduces the number of learnable model parameters, resulting in a lower computational cost of the training process. It should be noted that most modern CNN architectures do not use pooling layers, as they reduce tensor dimensionality without any learnable parameter and, therefore, cause a net loss of information within the network.

Chapter

Simulation setup and methodology 3

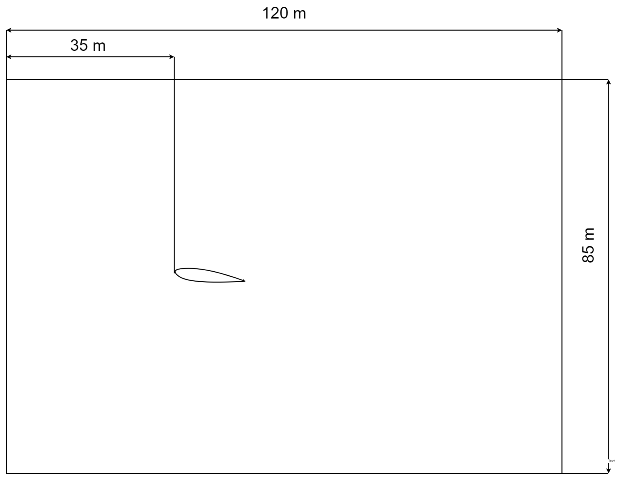


Fig. 3.1: Representation (not at scale) of the computational domain, with dimensioning.

The CFD simulation results used to train the proposed neural networks were obtained using the commercial CFD code ANSYS Fluent (see section 3.1). Due to the vast amount of data required to train the neural networks, a large number of simulations were necessary. Consequently, special attention was paid to reducing the runtime as much as possible while obtaining reasonably accurate simulations. Consequently, the analysis of high-accuracy 3D models such as DNS or LES simulations was considered unfeasible (in the case of DNS) or problematic due to the high computational cost and the resulting simulation time required to simulate hundreds of scenarios. It was therefore decided to use 2D RANS simulations as the best compromise between the time and computational cost constraints imposed by this work.

Preliminary simulations were carried out on common airfoil shapes. In addition, different combinations of domain shapes and meshing methods were considered: two of the domain shapes considered and two different meshing methods applied can be seen in Fig. 3.2 and 3.3. Detailed information on the final choice of domain shape and meshing method can be found in section 3.3.1. The proposed model consists of a rectangular-shaped domain, shown and dimensioned in Fig. 3.1, with a 70×110 m bounding box and discretized by a hybrid unstructured quad-dominant block-mesh. The airfoil adopted is the NASA SC(2)-0714, part of the NASA SC family of supercritical airfoils[70] and shown in Fig. 3.5. The airfoil, as can be seen in Fig. 3.5, has a chord equal to 1 m and is positioned in such a way that for a zero

Angle of Attack (AoA) the leading edge of the airfoil coincides with the origin of the system axes; for non-zero AoAs, the airfoil has been rotated around the point located at [0.25 m, 0 m], i.e. approximately around its neutral point.

The following sections explain in more detail the domain shape and the meshing method used, the physical model used and the physical properties of the fluid, the solver options, the data acquisition method, and, lastly, a sample of the results obtained from the carried out simulations is presented.

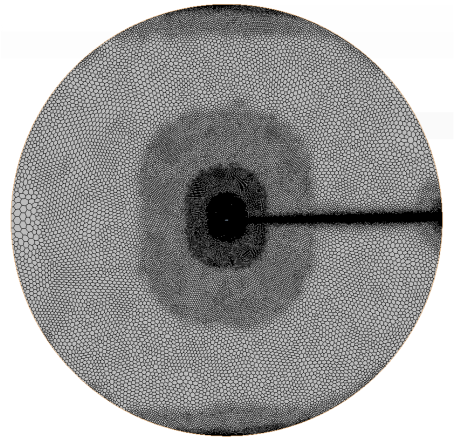


Fig. 3.2: One of the considered domain shapes, after mesh refinement. The circular domain was of excessive size in front of the airfoil, increasing the simulation runtime.

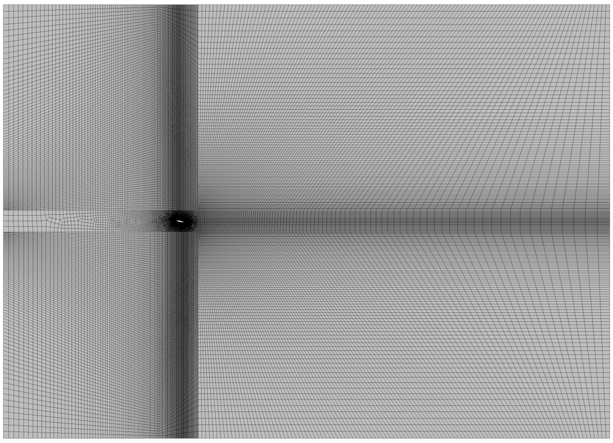


Fig. 3.3: One of the domain shapes initially considered, meshed using a semi-structured quad-predominant block mesh whose cells presented an excessive aspect ratio.

3.1 ANSYS Fluent

CFD simulations were carried out on ANSYS Fluent. It is a fluid dynamics simulation software developed by the American multinational company ANSYS, best known for its CFD analysis software. The model developed for the numerical simulations is carefully described in section 3.4 to allow the reproduction of the results presented here. The simulated systems were modelled locally on a PC thanks to the ANSYS Fluent license the Politecnico di Torino provided. The high numerical cost due to the accuracy and the number of simulation runs could not be managed on a personal computer. The simulations were performed remotely on the Politecnico di Torino High Performance Computing (HPC) clusters[71], to which I was kindly granted access to complete this work.

3.1.1 FVM

The CFD implementation of ANSYS Fluent is based on the FVM formulation of the Navier-Stokes equations. The Navier-Stokes equations are hyperbolic PDEs governing the conservation of fundamental physical properties over time (see section 2.1) and, as such, are conservation laws. The FVM is widely used in CFD as a means to discretize the set of PDEs, converting them into an algebraic set of equations[72].

The integral form of the equations serves as the foundation for the FVM. The integrals are discretized by dividing the physical domain into finite volumes called cells, generating a *mesh*[72,73]. The cell centre is typically assigned as a computational node where the relevant properties are calculated when solving the PDEs. The cell centroid value of a physical quantity (e.g. velocity) is usually defined as the average over the cell volume[72]. The conservation law is then written for each cell, and by the divergence theorem, any terms containing a divergence or gradient are written as a flux through the cell boundary. For example, given the eq. (3.1):

$$\frac{\partial(\rho Q)}{\partial t} + \frac{\partial\left(\rho u_j Q\right)}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\Gamma \frac{\partial Q}{\partial x_j}\right) - S_Q = 0 \tag{3.1}$$

This is the scalar convection-diffusion equation, a simplified equation that bears an obvious resemblance to the equation of conservation of momentum shown in eq. (2.8.b). Applying the divergence theorem, the integral form is as follows:

$$\int_\Omega \frac{\partial(\rho Q)}{\partial t} d\Omega + \int_{\partial\Omega} (\rho u_j Q) d(\partial\Omega) - \int_{\partial\Omega} (\Gamma \frac{\partial Q}{\partial x_j})(\partial\Omega) - \int_\Omega S_Q d\Omega = 0 \tag{3.2}$$

The spatial domain can be discretized by dividing it into cells. Assuming a one-dimensional domain for simplicity, the previous equation can be rewritten for each cell as:

$$\left(\frac{\partial\overline{\rho Q_i}}{\partial t} + \overline{S_{Q,i}}\right)\Omega_i + \sum_{k=i-1/2}^{i+1/2}\left(\left(\rho u_k Q_k - \Gamma_k \frac{\partial Q_k}{\partial x_k}\right)\partial\Omega_k\right) = 0 \tag{3.3}$$

In the equation shown above, the volume integral is approximated by multiplying the cell mean by the cell volume, and the sum of the fluxes at the cell boundary replaces the surface integral. The time discretization is introduced by dividing the physical time into discrete steps (*timesteps*).

The actual numerical scheme based on the FVM depends on both the spatial and the temporal discretization and, most critically, on the reconstruction/estimation method of the cell boundary fluxes. A schematization of an upwind FVM scheme

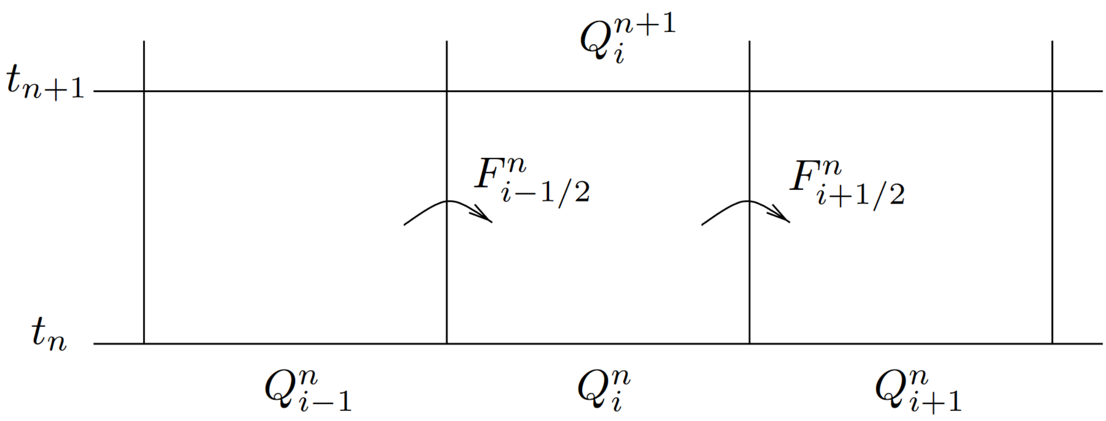


Fig. 3.4: Schematization of an upwind FVM scheme of a 1D conservation law shown in the space-time plane. Source: *LeVeque*, 2002[72].

can be seen in Fig. 3.4. The main benefit of the FV formulation is that these approaches are conservative because, as can be observed in eq. (3.3), the flux entering a given cell volume is the same as the flux leaving the adjacent volume. These approaches are then conservative not only at a local (cell) level but over the entire computational domain[30,72,73].

3.2 Parameters sweep

	AoA	Re	M
Values	3.0°	$5.000{\cdot}10^6$	0.500
	6.0°	$6.250{\cdot}10^6$	0.550
	9.0°	$7.500{\cdot}10^6$	0.600
	12.0°	$8.125{\cdot}10^6$	0.625
	13.5°	$8.750{\cdot}10^6$	0.650
	15.0°	$8.375{\cdot}10^6$	0.675
	16.5°	$1.000{\cdot}10^7$	0.700

Tab. 3.1: Summary of the free parameters of the simulated model and their respective values.

As shown in section 2.2, the body forces on a rigid body in a stationary freestream are essentially governed by the body geometry and the Mach number and Reynolds number of the freestream.

As mentioned above, the influence of the three parameters on the dimensionless airfoil parameters, on the wake, and on the near-field was explored by performing numerical simulations as the parameters varied. A parameter sweep was then performed, i.e. a simulation was carried out for each combination of parameter values considered. The parameters under consideration and their values are summarised in Tab. 3.1. For each freestream parameter, 7 different values were considered: the total number of scenarios considered is 343 (as $N_{sims} = V^P = 7^3 = 343$).

It is worth mentioning that the angle of attack obviously affects the mesh of the simulated model (see section 3.3.1). In addition, the Reynolds number affects the model mesh as the effect of the Reynolds number on the boundary layer thickness was considered when generating a prism layer on the airfoil surface (for more information, see section 3.3.1.1). In addition, the necessary boundary conditions were applied to each simulated scenario as the parameters varied. More information on the BCs used and the relevant considerations for flow field variable matching are explored in sections 3.4.5 and 3.4.5.3.

3.3 Model description

In the following subsections, the setup of the simulated models is described in detail, and the choices made are explained. The differences between the various simulated scenarios resulting from the parametrization of the physical system and the impact of the various free parameters on the mesh and the BCs are also highlighted.

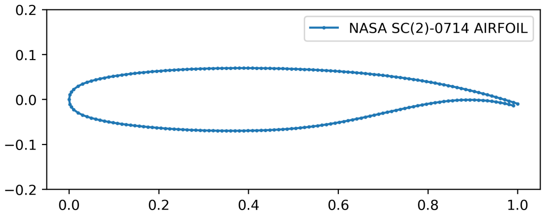


Fig. 3.5: Plot of NASA airfoil SC(2)-0714. Airfoil coordinates retrieved from http://airfoiltools.com/[74].

3.3.1 Mesh & domain shape

Mesh region	Edge size [m]	Growth Rate
Internal	10	1.2
Airfoil	0.004	1.015
Airfoil LE	0.001	1.015
Airfoil TE	0.001	1.015
Near-field	0.010	1.1
Wake	0.010	1.1
Freestream	10	1.2

Tab. 3.2: Summary of the principal mesh properties for each mesh region.

As mentioned above, different domain geometries and different domain spatial discretization were evaluated, as shown in Fig. 3.2 and 3.3. Ultimately, the chosen geometry is shown in Fig. 3.1, which also highlights the domain dimensions and airfoil positioning. The rectangular-shaped domain was ultimately chosen despite requiring more cells at equal domain size compared with other geometries (i.e. D-shaped, parabolic

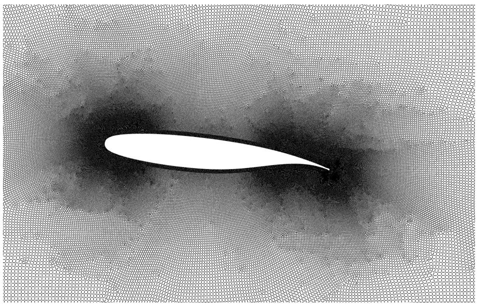


Fig. 3.6: Mesh near the airfoil before mesh adaptation. The mesh shown is referred to the α=6°, Re=10^7, M=0.7 scenario. Notice how the mesh around the trailing and leading edges is finer.

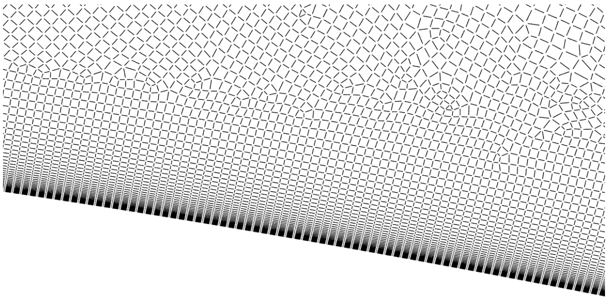


Fig. 3.7: Close up of the prism layer and the surrounding quad/tria hybrid mesh.

shape, etc.)[75]. This is because it allowed the generation of a high-quality, high-orthogonality, unstructured non-conformal mesh. The domain size, with a bounding box of 120 m×85 m, is such that it allows the pressure freestream condition to be met at the boundaries and improves the accuracy of the data produced, especially at high AoAs[76].

Initially, the use of a structured mesh was considered. The advantage of this is that the algebraic system of equations generated by the solver is diagonally banded: this allows the use of low-cost matrix inversion methods that allow a less computationally intensive resolution of the algebraic systems[40,75]. However, the inability to perform adaptive refinement of structured meshes using ANSYS Fluent led to the choice of using an unstructured, non-conformal mesh. The ability of unstructured meshes to adapt to complex geometries and cluster points in regions of high gradient and curvature is another driving factor[40,75]. This feature was considered critical since the parametric sweep required multiple meshes to be automatically generated without direct oversight: structured or conformal unstructured meshes suffered from low-quality elements or poorly formed prism layers for some of the scenarios considered. In comparison, unstructured non-conformal meshes with the ability to cluster cells at high curvature points allowed the generation of meshes of higher quality.

As explained in section 3.2, the parameter sweep encompassed 343 unique scenarios. Of the three free parameters considered (AoA, Re, M), only AoA and Re have an a priori known effect on the model mesh: AoA because it changes the orientation of the airfoil, and Re because it governs the thickness of the boundary layer (see section 3.3.1.1). Therefore, 49 different meshes were automatically generated, one for each unique combination of the examined values of AoA and Re. During the

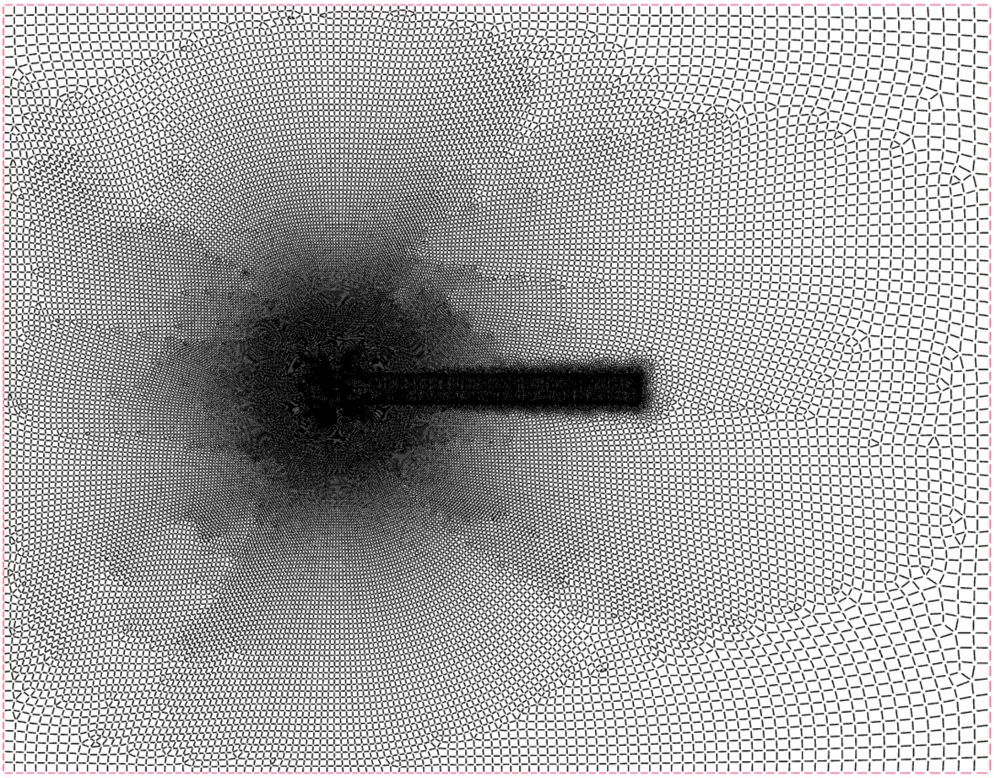


Fig. 3.8: Mesh before adaptation. The mesh shown is referred to the α=6°, Re=10^7, M=0.7 scenario. Notice how the mesh around the airfoil is finer and how the wake mesh is finer.

simulations, the unique conditions of the considered scenario, including the Mach number of the freestream and their influence on the mesh, were taken into account by implementing adaptive mesh refinement (see section 3.3.1.2).

The resulting meshes contained an average of approximately 600,000 cells prior to any adaptive refinement. As can be seen, the mesh on the airfoil is significantly finer, with the average cell size being ~1 mm (before adaptation; minimum cell size after adaptation ~0.2 mm). In addition, it was ensured that the near-field mesh was fine enough to resolve the flow, resulting in an average near-field cell size of about 8 mm. A wake refinement was also performed, enforcing an average cell size of approximately 1 cm. A summary of the main mesh properties for each mesh region is given in Tab. 3.2.

3.3.1.1 Prism Layer

In most practical flows, walls and barriers are a source of vorticity[40,47]. Therefore, accurate prediction of flow and turbulence characteristics throughout the wall boundary layer is crucial. To capture the significant transverse gradients (such as velocity and temperature) within the boundary layer, prism cells were generated by extruding the airfoil surface. A detail of the mesh prism layer is shown in Fig. 3.7. The flow-oriented high aspect ratio cells allow a high enough resolution without an excessively fine mesh in the near-wall region, allowing a reduced mesh cell count. Fig. 3.7 shows how the length and growth rate of the prism layer have been tuned to ensure a smooth transition between the prism layer and the quad-dominant mesh.

As highlighted in section 3.4.4, the prism layer was set to be 55 cells high to resolve the transverse gradient[48] accurately. In addition, as recommended in the user manual[48,77], the cell growth rate was set to 1.1 (see section 3.4.4). The number of cells and the growth rate were chosen to ensure a first cell y$^+$ of ~0.5 (see section 3.4.4). An ad-hoc mesh adaptation criterion was applied to the prism layer to ensure a first-cell y$^+$ in the range of 0.1-10 to avoid excessive coarseness or excessive fineness of the prism layer (see section 3.3.1.2).

Quantity	Value
Number of cells	55
Growth factor	1.1
Total Thickness	f(Re)
Avg. first cell y$^+$	~1

Tab. 3.3: Prism layer settings.

The prism layer thickness was determined by estimating the boundary layer thickness. Estimating the boundary layer thickness for a compressible turbulent flow is not trivial as the data in literature is lacking[47]. A rough estimate was obtained from the results published by Stratford et al. (1959)[78] on the compressible turbulent BL over a flat plate. For a freestream Mach-adjusted Reynolds number Re$_X$ the Boundary Layer (BL) thickness δ is:

$$\delta(X) = 0.37XRe_X^{-1/5} \qquad \text{For Re}_X\text{~}10^6 \qquad (3.4)$$

$$\delta(X) = 0.23XRe_X^{-1/6} \qquad \text{For Re}_X\text{~}10^7 \qquad (3.5)$$

Where P is the parameter describing the dependence on the Mach number, X is the integral average of the wall position over the parameter P

$$P = \left[M/\left(1+M^2/5\right)\right]^4 \qquad (3.6)$$

$$X = P^{-1}\int_0^x Pdx \qquad (3.7)$$

and the freestream Mach-adjusted Reynolds number is defined as $Re_X = \frac{UX}{\nu}$.

It is worth remarking that eq. (3.4) has the same structure as the well-known Prandtl expression for a turbulent incompressible boundary layer in a flat channel[79], that is:

$$\delta(x) \approx 0.37xRe_x^{-1/5} \tag{3.8}$$

The Mach number distribution over the airfoil must be known in order to calculate the factor P shown in eq. (3.6), necessitating a second simulation to estimate the distribution. This was deemed costly and unnecessary and, as experimental findings imply that at moderate Mach numbers, the boundary layer thickness is nearly independent of Mach[78] (as can be seen in Fig. 3.9), it was opted to assume that X≈x. Moreover, as the minimum simulated Reynolds number is $5{\cdot}10^6$, eq. (3.5) (plotted in Fig. 3.10) was deemed the most representative of the actual BL thickness. The Mach number distribution over the airfoil must be known in order to calculate the factor P shown in eq. (3.6), requiring a preliminary simulation to estimate the distribution. This was considered costly and unnecessary, and since experimental results suggest that at moderate Mach numbers the boundary layer thickness is nearly independent of Mach[78] (as can be seen in Fig. 3.9), it was opted to assume that X≈x. Furthermore, since the minimum simulated Reynolds number is $5{\cdot}10^6$ eq. (3.5) (plotted in Fig. 3.10) was deemed most representative of the actual BL thickness.

The prism layer δ_{prism} was then set to $1.2{\cdot}\delta(L)$ to ensure that the entire boundary layer was resolved (see section 3.4.4). The aforementioned prism layer characteristics are summed in Tab. 3.3. It should be mentioned that, as the prism layer thickness is Reynolds-dependent, every case scenario at different Reynolds resulted in a different mesh (as previously explained in section 3.2).

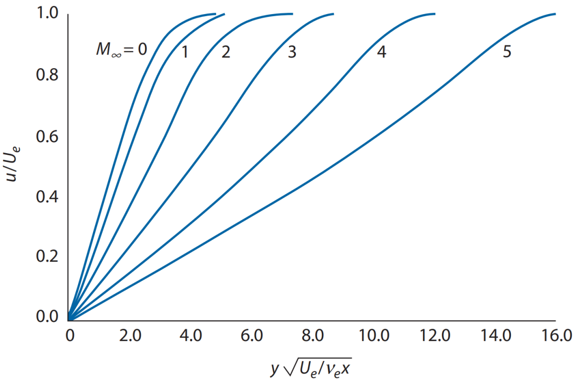


Fig. 3.9: Velocity profiles and boundary layer heights on an adiabatic flat plate for Prandtl number Pr=0.75 and laminar flow. Source: *Rodney D. W. Bowersox*, 2011[47].

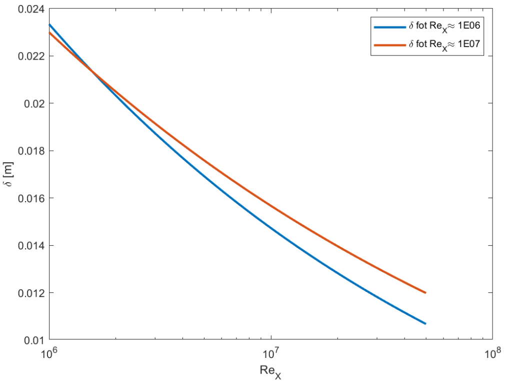


Fig. 3.10: Turbulent compressible boundary layer thickness δ as a function of the Mach-adjusted Reynolds number Re_X.

3.3.1.2 Mesh Adaptation

Option	Value
Maximum refinements per cell	3
Minimum cell size	0.0002 [m]

Tab. 3.4: Options of the mesh adaptation algorithm.

As previously mentioned, a new mesh was generated AoA) and Reynolds Number. However, an ideal mesh for every simulated scenario depends on both the Mach number and the presence of shocks, transition regions, flow separations, and other flow features, which cannot be accurately predicted during the mesh generation. By employing solution-adaptive refinement, cells can be added where needed, improving the flow field's feature resolution and aiding accuracy and convergence rate[48,77]. As the flow solution itself is used to identify where additional cells should be placed, the resulting mesh should be well-suited for the investigated system[48,77]. Ideally, no computational resources are wasted by including unnecessary cells.

During mesh adaptation, cells are flagged to be refined or coarsened according to an adaptation function and then adapted via the hanging-node method. The adaptation function has been implemented using *cell registers*, a tool for marking cells according to specific user-defined criteria. Four user-defined criteria were defined, based on 4 scalar field variables: ρ, p, T and $|\mathbf{V}|_2$ (that is, the Euclidean norm of the velocity vector). This criterion was derived using the *gradient approach* (recommended with problems with strong shocks[77]). In this method, the Euclidean norm of the gradient of selected flow field variables is multiplied by a distinctive length scale (which in 2D corresponds to the cell area). The gradient function in 2D takes the following form:

$$|e_i| = (A_{\text{cell}})^{\frac{1}{2}} \, |\nabla \varphi_i| \tag{3.9}$$

Where φ_i is the generic field variable value on the cell i and e_i is the function value over the cell i. The resulting functions are normalized by their average value over the domain.

$$\varepsilon_i = \frac{|e_i|}{\overline{|e|}} \tag{3.10}$$

The 4 functions obtained were used to define 4 cell registers to be used for adaptation refinement and 4 cell registers to be used for adaptation coarsening, as summarised in Tab. 3.4. The cell registers were used to construct the mesh adaptation criteria, shown in eq. (3.11), boolean conditions governing the automatic mesh refinement.

$$\left(\nabla\rho_{Reg.} \wedge \nabla p_{Reg.} \wedge \nabla|\boldsymbol{V}|_{Reg} \wedge REF_{count.}\right) \vee \left(\nabla\rho_{Reg.} \wedge \nabla T_{Reg.} \wedge \nabla|\boldsymbol{V}|_{Reg} \wedge REF_{count.}\right) \vee \\ \vee \left(\nabla p_{Reg.} \wedge \nabla T_{Reg.} \wedge \nabla|\boldsymbol{V}|_{Reg} \wedge REF_{count.}\right) \vee \left(\nabla\rho_{Reg.} \wedge \nabla p_{Reg.} \wedge \nabla T_{Reg.} \wedge REF_{count.}\right) \tag{3.11}$$

Where:

- $\vee$ is the boolean OR operator.
- $\wedge$ is the boolean AND operator.
- $\nabla\rho_{Reg.}$ is the cell register boolean value related to the density gradient on the cells, averaged over the average domain gradient value.
- $\nabla p_{Reg.}$ is the cell register boolean value related to the gauge pressure gradient on the cells, averaged over the average domain gradient value.
- $\nabla T_{Reg.}$ is the cell register boolean value related to the static temperature gradient on the cells, averaged over the average domain gradient value.
- $\mathrm{REF}_{\mathrm{count.}}$ is the refinement number boolean condition, which returns true if the cell has been adapted fewer times than the maximum number of refinements allowed (i.e. 3 cell adaptations). It can be expressed as $\mathrm{N}_{\mathrm{ref.}}$<3.

The cells were flagged to be refined in cells where the adaptation function returned a True value.

The boolean condition governing the automatic mesh coarsening, built from the coarsening cell registers summed up in Tab. 3.4, is shown in eq. (3.12).

$$\nabla\rho_{Reg.} \wedge \nabla p_{Reg.} \wedge \nabla T_{Reg.} \wedge \nabla|\boldsymbol{V}|_{Reg} \tag{3.12}$$

The cells were flagged to be coarsened in cells where the coarsening function returned a True value. Once the cells are marked for adaptation, the mesh is adapted via the hanging node method[48]. The method is named after the fact that the meshes it produces are distinguished by nodes on edges and faces (known as *hanging nodes*), generated by splitting cells. An example of the hanging node method is shown in Fig. 3.11. To ensure accuracy, neighbouring cells cannot differ by more than one level of refinement: this prevents the adaptation from causing excessive variations in cell volume and ensures that the positions of the parent (original) and child (refined) cell centroids are identical (reducing errors in the flux evaluations)[48].

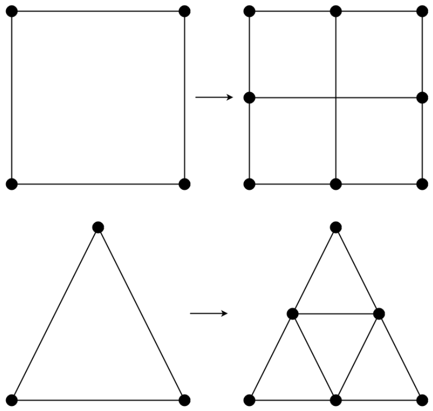


Fig. 3.11: Examples of cell refinement via the hanging node method.

Cell register	Threshold value
$(\nabla \rho)_{\text{Reg., coars.}}$	<0.3
$(\nabla p)_{\text{Reg., coars.}}$	<0.3
$(\nabla T)_{\text{Reg., coars.}}$	<0.3
$(\nabla \lvert V \rvert)_{\text{Reg., coars.}}$	<0.3
$(\nabla \rho)_{\text{Reg., ref.}}$	>5
$(\nabla p)_{\text{Reg., ref.}}$	>4
$(\nabla T)_{\text{Reg., ref.}}$	>4
$(\nabla \lvert V \rvert)_{\text{Reg., ref.}}$	>5

Table 3.5: Gradient-based cell registers and relative activation thresholds.

3.4 Physical model

This section discusses the selected physical models. The chosen models are summarised in Tab. 3.6. The following subsections describe the selected models in more detail if deemed necessary.

As already mentioned, the simulated system is two-dimensional and turbulent. The free stream is assumed to be fully developed, i.e. the flow is already transitioned (this corresponds to an intermittency γ=1). The physics of the phenomena is expected to be unsteady, especially at high AoAs where processes such as separation and reattachment are expected (see section 2.4.2). Due to the range of Mach numbers simulated and the compressible flow regime, adopting a coupled energy model was deemed necessary, as recommended in the manual[48]. The coupled density-based model simultaneously solves the governing equations for momentum, continuity and (as the coupled flow option was enabled) energy. Th equations for additional scalars (SST k-ω model equations, γ-Re_θ equations) are solved sequentially (i.e. separately from each other and from the coupled set).

3.4.1 Fluid Properties

In Tab. 3.7, the physical properties of the fluid are shown. The working fluid, air, is considered to be an ideal gas. The molecular weight has been set as the constant for dry air, i.e. a mixture of nitrogen, oxygen, hydrogen, argon and other trace

Models and general settings	Description/Notes
Gas	See section 3.4.1
Ideal Gas	See section 3.4.1
Transient	
Coupled Flow	
Turbulent	
Two Dimensional	
Operating Pressure	0 Pa
Density Based solver	
Reynolds-Averaged Navier-Stokes	See section 3.4.2
SAS-SST (Menter) k-ω Turbulence	See section 3.4.2.3
γ-Re_θ Transition	See section 3.4.3.1
y^+-Independent Wall Treatment	See section 3.4.4

Tab. 3.6: Summary of the selected physical models and general settings.

gases. The molecular weight can be set as a constant as the fluid is considered non-reactive. The specific heat is assumed to be constant: this is in agreement with what was established in section 2.1, namely that the fluid can be considered *calorically perfect* in the temperature range considered.

The dynamic viscosity and the thermal conductivity of the fluid have been set as variable, following Sutherland's Law, according to:

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0}\right)^{3/2} \frac{T_0 + S}{T + S} \quad \mu_0 = 1.716 \cdot 10^{-5}\,\mathrm{Pa\,s} \quad T_0 = 273.15\,K \quad S = 110.4\,K \tag{3.13}$$

Material Properties	Value
Dynamic Viscosity	f(Sutherland's Law)
Molecular Weight	28.9664 kg kmol^{-1}
Specific Heat	1003.62 J kg^{-1} K^{-1}
Thermal Conductivity	f(Sutherland's Law)

Tab. 3.7: Material properties of the fluid.

$$\frac{\kappa}{\kappa_0} = \left(\frac{T}{T_0}\right)^{3/2} \frac{T_0 + S_\mathrm{k}}{T + S_\mathrm{k}} \qquad \kappa_0 = 0.02414\,\mathrm{W\,m^{-1}\,K^{-1}} \qquad T_0 = 273.15\,K \qquad S_\kappa = 194.0\,K \tag{3.14}$$

It should be noted, however, that the freestream dynamic viscosity and the freestream thermal conductivity are set as constant since for each considered variation of the freestream Mach and Reynolds numbers, the freestream temperature is set to a constant value of T_∞=293.15 K. Therefore, the Prandtl number of the freestream is set as a constant.

$$Pr_\infty = \frac{c_p \mu(T_\infty)}{\kappa(T_\infty)} = 0.6989 \tag{3.15}$$

For more information on the topic, see section 3.4.5.3.

3.4.2 RANS & Turbulence modelling

Turbulence is an inherently tridimensional phenomenon. As such, it can be argued that 2D CFD simulations, which will be used to train the neural network (see Chapter 4), cannot calculate an accurate flow field. Moreover, we cannot simulate a fully 2D flow as 2D fluid dynamics and 2D turbulence present radically different properties, such as the absence of the *vortex stretching and tilting* mechanism, which causes local instabilities in the flow and the subsequent breakup of the vortices into smaller ones. In fact, from the vorticity equation (eq. (2.14)) it can be shown that, in a 2D fluid, as the velocity field **u**(x,y) only has 2 components (and obviously holds no dependence on the x_3 direction, so that any partial derivative on that direction equals to zero) then the vorticity vector only has 1 component (to be more precise $\boldsymbol{\omega}$=(0, 0, $\frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2}$). The vortex stretching-tilting term is equal to 0 as

$$\omega_j \frac{\partial u_i}{\partial x_j} \quad \Rightarrow \quad \omega_3 \left(\frac{\partial u_1}{\partial x_3} + \frac{\partial u_2}{\partial x_3}\right) = \left(\frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_3}\right)\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_3}\right) = 0 \tag{3.16}$$

In the absence of the stretching-tilting term, the energy cascade cannot be sustained[38]. In these conditions, the vorticity's material derivative is null, so each particle retains its initial vorticity (ignoring the aid of viscous diffusion, which tends to spread the vorticity out and make it uniform and the effect due to the compressibility and baroclinic terms in eq. (2.14)). Moreover, it can be shown (and it has been done by DNS of pure 2D flows[80]) that the breakup of turbulent structures into smaller ones does not occur. Instead, smaller turbulent structures tend to coalesce into bigger ones, an obvious non-physical behaviour[80].

A pseudo-2D simulation can be achieved by RANS turbulence modelling. The RANS formulation is obtained by, in its most general form, by ensemble average of the Navier-Stokes equations[30], where the ensemble average in defined as:

$$\langle x(t_1)\rangle = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} x^{(i)}(t_1) \tag{3.17}$$

And represent the average of the time-dependent quantity x over N realizations of the same phenomena (e.g. wind tunnel experiments). The RANS equations are obtained by applying the average operator in the Navier-Stokes equation. For example, in eq. (3.19), the RANS for an incompressible flow are shown.

$$\frac{\partial\langle u_i\rangle}{\partial x_i} = 0 \tag{3.18}$$

$$\rho\left(\frac{\partial\langle u_i\rangle}{\partial t} + \frac{\partial}{\partial x_j}\left\langle u_i u_j\right\rangle\right) = -\frac{\partial\langle p\rangle}{\partial x_i} + \mu\nabla^2\langle u_i\rangle + \rho g_i \tag{3.19}$$

However, the averaged quantities can be decomposed as

$$\left\langle u_i u_j\right\rangle = \langle u_i\rangle\left\langle u_j\right\rangle + \left\langle u_i' u_j'\right\rangle \tag{3.20}$$

Resulting in the following expression of the averaged conservation of momentum for an incompressible fluid.

$$\rho\left(\frac{\partial\langle u_i\rangle}{\partial t} + \frac{\partial}{\partial x_j}\left(\langle u_i\rangle\left\langle u_j\right\rangle\right)\right) = -\frac{\partial\langle p\rangle}{\partial x_i} + \mu\nabla^2\langle u_i\rangle + \frac{\partial}{\partial x_j}\left(-\rho\left\langle u_i' u_j'\right\rangle\right) + \rho g_i \tag{3.21}$$

The RANS formulation allows the simulation of 2D turbulent flows as the equations describe the behaviour of the average flow. As such a 2D turbulent can be described by the RANS equations as a flow with a zero average velocity in one direction (for example x$_3$) and no dependence on that direction (such that every derivative on that direction is equal to 0 and $\frac{\partial(\cdot)}{x_3} = 0$). It can be observed that the linear terms in the NS equations are left untouched. However, the average operation applied to the nonlinear terms produced additional terms. For example, the incompressible equation of conservation of momentum in the RANS formulation shown in eq. (3.21) is characterized by the presence of an additional term: the divergence of the tensor $R_{ij} = \rho\left\langle u_i' u_j'\right\rangle$. This tensor, known as the *Reynolds stresses tensor*, can be interpreted as the convection of momentum due to turbulent motion.

The RANS equations are solved for the average flow velocities $\langle U_i\rangle$. Consequently, the presence of the Reynolds Stresses gives rise to the so-called *closure problem of* *turbulence*[38,79] due to additional unknowns in the equations. A *turbulence model*, which correlates the average flow to the Reynolds stress tensor, is needed to close the equation system and numerically solve it. The choice of the appropriate turbulence model is discussed in section 3.4.2.3.

3.4.2.1 Favre averaging

Applying the ensemble average operator (shown in eq. (3.17)) to the incompressible Navier-Stokes equations will produce additional new terms due to the presence of nonlinear terms. While in the incompressible formulation, the only new term produced was the Reynolds Stress tensor, in the compressible formulation, the continuity equation presents an additional nonlinear term:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i} \underbrace{(\rho u_i)}_{\text{Nonlinear term}} = 0 \tag{3.22}$$

Therefore, the continuity equation would produce the Reynolds stresses term due to the $\langle \rho u_i' \rangle$ average. Moreover, averaging the compressible conservation of momentum equation would produce an additional term due to the decomposition (as shown in eq. (3.20)) of the term $\langle \rho u_i' u_j' \rangle$. This further complicates the RANS formulation for a compressible flow. However, as shown by Morkovin (1962)[81], even at supersonic speed, the turbulence perturbations caused by turbulence are small. The hypothesis holds if there is no significant heath conduction in the flow or if the fluid can be considered chemically inert (e.g. absence of combustion). In such conditions, the ensemble average can be substituted by the Favre average, also known as *mass average*, shown in eq. (3.23).

$$\tilde{u}_i = \frac{1}{\bar{\rho}} \lim_{T \to \infty} \frac{1}{T} \int_t^{t+T} \rho(\mathrm{x}, \tau) u_i(\mathrm{x}, \tau) d\tau \tag{3.23}$$

Which can be rewritten as:

$$\tilde{u_i} = \frac{\langle \rho u_i \rangle}{\langle \rho \rangle} \tag{3.24}$$

So that the nonlinear terms in the compressible Navier-Stokes equations can be rewritten as:

$$\langle \rho u_i \rangle = \langle \rho \rangle \widetilde{u_i} \Rightarrow -\langle \rho \rangle \widetilde{u_i'} \tag{3.25}$$

$$\left\langle \rho u_u u_j \right\rangle = \langle \rho \rangle \widetilde{u_i u_j} \Rightarrow -\langle \rho \rangle \widetilde{u_i' u_j'} \tag{3.26}$$

From now on, to improve the notation readability, the Favre average and the ensemble average will be represented by the overline symbol $\overline{(\cdot)}$. Moreover, it should be noted that

$$\overline{\rho}\tilde{u}_i = \overline{\rho u_i} + \overline{\rho' u_i'} \tag{3.27}$$

Where the ' superscript denotes the fluctuating component of the ensemble average. Similarly, the Favre decomposition is written as:

$$u_i = \tilde{u}_i + u_i'' \tag{3.28}$$

It follows that:

$$\overline{\rho u_i} = \bar{\rho}\tilde{u}_i + \overline{\rho u_i''} \tag{3.29}$$

More detailed information on the Favre average and its application can be found in Wilcox (2006)[30].

3.4.2.2 Compressible RANS equations

By applying eq. (3.23) to the compressible Navier-Stokes equations for a Newtonian Fourier ideal gas (eq. (2.8)) the RANS compressible equations can be obtained.

$$\begin{cases} \dfrac{\partial \bar{\rho}}{\partial t} + \dfrac{\partial}{\partial x_i}\left(\bar{\rho}\tilde{u}_i\right) = 0 & \text{(3.30.a)} \\ \dfrac{\partial}{\partial t}\left(\bar{\rho}\tilde{u}_i\right) + \dfrac{\partial}{\partial x_j}\left(\bar{\rho}\tilde{u}_j\tilde{u}_i\right) = -\dfrac{\partial p}{\partial x_i} + \dfrac{\partial}{\partial x_j}\left[\bar{\tau}_{ji} - \overline{\rho u_j'' u_i''}\right] & \text{(3.30.b)} \\ \dfrac{\partial}{\partial t}\left[\bar{\rho}\left(\tilde{e} + \dfrac{\tilde{u}_i\tilde{u}_i}{2}\right) + \dfrac{\overline{\rho u_i'' u_i''}}{2}\right] + \dfrac{\partial}{\partial x_j}\left[\bar{\rho}\tilde{u}_j\left(\tilde{h} + \dfrac{\tilde{u}_i\tilde{u}_i}{2}\right) + \tilde{u}_j\dfrac{\overline{\rho u_i'' u_i''}}{2}\right] = & \text{(3.30.c)} \\ = \dfrac{\partial}{\partial x_j}\left[-q_{L_j} - \overline{\rho u_j'' h''} + \overline{\tau_{ji} u_i''} - \overline{\rho u_j'' \dfrac{1}{2} u_i'' u_i''}\right] + \dfrac{\partial}{\partial x_j}\left[\tilde{u}_i\left(\bar{\tau}_{ij} - \overline{\rho u_i'' u_j''}\right)\right] & \text{(3.30.d)} \end{cases}$$

It should be mentioned that the equations in the system shown in eq. (3.30) are identical to their laminar counterpart except for the presence of the Favre-averaged and ensemble-averaged terms.

3.4.2.3 SAS-SST k-ω model

The chosen turbulence model to achieve closure of the system shown in eq. (3.30) is the SAS-SST k-ω model. It is a modification of the Shear Stress Transport turbulence model introduced by Menter (1994)[82]. The SAS-SST model is an eddy-viscosity model which employs the so-called Boussinesq approximation winch correlates the Reynolds stresses to the *eddy viscosity* of the flow (a measure of the additional flow diffusivity due to the chaotic, turbulent motion) to the average velocity field of the flow. It is a two-equation model which combines the widely used k-ϵ and k-ω turbulence models: more specifically, the SAS formulation is a modification of the classical RANS SST formulation that allows the resolution of smaller-scale turbulence, resulting in improved accuracy of numerical results[48].

$$-\overline{u_i' u_j'} = \nu_t \left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{3.31}$$

The additional eddy viscosity term ν_t has to be determined. By means of dimensional analysis, it can be shown that $\nu_t = \frac{k}{\omega}$[84]1, where k is the turbulent kinetic energy and ω is the kinetic energy specific dissipation rate. One fundamental difference between the molecular viscosity ν and the eddy viscosity ν_t is that ν_t depends on the fluid properties and the flow field structure. Thus, the value of ν_t over the whole field needs to be calculated. The SAS-SST model's modified κ-ω transport equation is shown in eq. (3.32) and (3.34)[82].

$$\frac{\partial \rho k}{\partial t} + \frac{\partial}{\partial x_i} \left(\rho u_i k \right) = G_k - \rho c_\mu k \omega + \frac{\partial}{\partial x_j} \left[\left(\mu + \frac{\mu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] \tag{3.32}$$

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial}{\partial x_i} \left(\rho u_i \omega \right) = \alpha \frac{\omega}{k} G_k - \rho \beta \omega^2 + Q_{SAS} + \frac{\partial}{\partial x_i} \left[\left(\mu + \frac{\mu_t}{\sigma_\omega} \right) \frac{\partial \omega}{\partial x_j} \right] \tag{3.33}$$

$$\underbrace{+ \left(1 - F_1 \right) \frac{2\rho}{\sigma_{\omega,2}} \frac{1}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}}_{\text{Cross-diffusion term}} \tag{3.34}$$

The turbulent kinetic energy can be directly obtained from the RANS conservation of momentum equation (eq. (3.30)). The derivation is not shown as it is beyond the scope of this work. The ω equation is usually postulated in k-ω models, and, indeed, most of them differ on nothing but the structure of the ω equation. The SST model itself differs from the standard k-ω model by the addition of a cross-diffusion term in the ω equation (shown in eq. (3.34)) and by the usage of different modelling constants shown in eq. (3.35). While the omega quantity transport equation in the SST model is postulated from the basic mechanical processes characterizing the flow (i.e. unsteadiness, convection, production, dissipation, etc.) according to a heuristic methodology, the SAS-SST formulation uses the formulation proposed by Rotta (1968, 1972). In general, both equations are adapted by adopting additional terms, the value of which is obtained semi-empirically and based on the turbulence length scale.

[2] Formally, the formula is valid only for incompressible flows. However, due to the Morkovin's hypothesis, it holds true for moderate Mach numbers.

The family of k-ω models have been widely used, and it's proven to be accurate in the near wall for attached boundary layers and in mildly separated flows[30,85]. Moreover, it provides good prevision even in the presence of shocks and shocks-induced separations[30]. Even the standard k-ω model has a sharp sensitivity to the freestream value of ω and gives less accurate results in *free shear flows* such as jets, stratified flows or wake[30,82,85], the SAS-SST model overcomes such deficiencies.

$$F_2 = \tanh\left[\left[\max\left(\frac{2\sqrt{k}}{\beta^{*}\omega y}, \frac{500\nu}{y^2\omega}\right)\right]^2\right] \tag{3.35.a}$$

$$P_k = \min\left(\tau_{ij}\frac{\partial U_i}{\partial x_j}, 10\beta^{*}k\omega\right) \tag{3.35.b}$$

$$F_1 = \tanh\left\{\left\{\min\left[\max\left(\frac{\sqrt{k}}{\beta^{*}\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\sigma_{\omega 2}k}{CD_{k\omega}y^2}\right]\right\}^4\right\} \tag{3.35.c}$$

$$CD_{k\omega} = \max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-10}\right) \tag{3.35.d}$$

$$\phi = \phi_1 F_1 + \phi_2\left(1 - F_1\right) \tag{3.35.e}$$

$$\alpha_1 = \frac{5}{9}, \quad \alpha_2 = 0.44 \tag{3.35.f}$$

$$\beta_1 = \frac{3}{40}, \quad \beta_2 = 0.0828 \tag{3.35.g}$$

$$\beta^{*} = \frac{9}{100} \tag{3.35.h}$$

$$\sigma_{k1} = 0.85, \quad \sigma_{k2} = 1 \tag{3.35.i}$$

$$\sigma_{\omega 1} = 0.5, \quad \sigma_{\omega 2} = 0.856 \tag{3.35.j}$$

The model mentioned above is implemented, in ANSYS Fluent, as the *SAS-4 equation Transition SST model* as it includes the necessary corrections to include the γ-Re$_\theta$ transition model quantities γ and Re$_\theta$.

3.4.3 Transition

As highlighted in section 2.4, the boundary layer is expected to transition near the leading edge due to the high Reynolds number. Additionally, for high AoA, the BL is expected to separate (stalling the airfoil) and, eventually, reattach to the surface, producing recirculation bubbles over the airfoil upper surface. Additionally, at high AoA, the boundary layer is expected to detach on a periodic basis, producing a von Kármán vortex sheet, essentially behaving like a buff body. However, the boundary layer separation region depends on the BL regime. That is, turbulent boundary layers detach later than laminar ones[40,47] (see section 2.4.2). Due to the compressible flow regime and high Reynolds number of the freestream, other complex flow behaviours are expected, such as *shock-induced separations*, consequent reattachment of the boundary flow and later transition of the reattached boundary layer.

While the sparse flow field data used to train the neural network (Chapter 4) is measured in the wake far from the body surface, the wake structure depends on the separation point and the transition region. This required the implementation of an additional transition model. The chosen model was the γ-Re$_\theta$ model.

3.4.3.1 γ-Re$_\theta$ transition model

The γ-Re$_\theta$ model is a two-equation transition model based on the coupling of the aforementioned SST k-ω turbulence model with the transport equations of two additional scalar quantities: the flow intermittency γ (see section 2.4) and the transition momentum thickness Reynolds number $\widetilde{Re}_{\theta t}$ (the momentum thickness Reynolds number at the point of transition)[86]. The two equations are shown below.

$$\frac{\partial(\rho\gamma)}{\partial t} + \frac{\partial\left(\rho u_j \gamma\right)}{\partial x_j} = P_{\gamma 1} - E_{\gamma 1} + P_{\gamma 2} - E_{\gamma 2} + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_f}\right)\frac{\partial\gamma}{\partial x_j}\right] \tag{3.36}$$

$$\frac{\partial\left(\rho\widetilde{Re}_{\theta t}\right)}{\partial t} + \frac{\partial\left(\rho u_j \widetilde{Re}_{\theta t}\right)}{\partial x_j} = P_{\theta t} + \frac{\partial}{\partial x_j}\left[\sigma_{\theta t}\left(\mu + \mu_t\right)\frac{\partial\widetilde{Re}_{\theta t}}{\partial x_j}\right] \tag{3.37}$$

Eq. (3.36) is the transport equation for intermittency, which regulates the production and dissipation of turbulent kinetic energy k in the boundary layer.

Eq. (3.37) is the transition momentum thickness Reynolds number transport equation. The momentum thickness Reynolds number at transition is considered to be dependent on the freestream flow conditions such as turbulence, fluid properties, etc.[86] (see eq. (3.39)). However, the turbulence properties and the Re$_{\theta\,\mathrm{t}}$ value can change across the whole domain. The transition model then estimates the freestream Re$_{\theta\,\mathrm{t}}$ value and treats Re$_{\theta\mathrm{t}}$ as an additional transported scalar which can be convected and diffused into the boundary layer[48,86]. The momentum thickness Re$_\theta$ is defined as rescaled vorticity Reynolds number, that is:

$$Re_\theta = \frac{Re_{\omega\,\max}}{2.193} = \frac{\frac{\rho y^2}{\mu}\frac{\partial u}{\partial y}}{2.193} \tag{3.38}$$

The vorticity Reynolds number shown in eq. (3.38) is a local property easily computed as it is a simple function of density, viscosity and vorticity.

Both eq. (3.36) and (3.37) include unsteady, convection, dissipation/diffusion terms and production/destruction terms. Those are not expanded upon so as not to deviate from the scope of the present work.

It is important to note that the suggested transport equations are postulated (just as the SST model ω transport equation is) and serve as a foundation for the incorporation of experimentally derived correlations into general-purpose CFD methods and do not make any attempt at simulating the physics of the transition process[86].

The model employs 3 empirical correlations, whose values must be experimentally measured[48,86]. They are shown in eq. (3.39).

$$\begin{aligned} Re_{\theta t} &= f(Tu,\lambda) \\ F_{\text{length}} &= f\left(Re_{\theta t}\right) \\ Re_{\theta c} &= f\left(\tilde{e}_{\theta t}\right) \end{aligned} \tag{3.39}$$

$\mathrm{Re}_{\theta \mathrm{t}}$ is the Reynolds number value at the place where the velocity profile first departs from a purely laminar profile, whereas $\mathrm{Re}_{\theta \mathrm{c}}$ is the Reynolds number value at the location when turbulence first begins to develop. $\mathrm{F}_{\text{length}}$ is the distance from the stagnation points at which the transition process starts.

The empirical correlations in eq. (3.39), obtained from numerical tuning of empirical measurements

The functional relationship of the empirical correlations shown in eq. (3.39) was obtained from the results published by Malan et al. (2009)[87] for the calibration of the γ-Re_θ model for commercial CFD codes. The above relations were obtained from numerical tuning of empirical measurements[87]. The empirical correlations are shown in eq. (3.40).

$$\begin{aligned} Re_{\theta t} = Re_{\theta c} &= \min\left[0.615\cdot \widetilde{\mathrm{Re}_{\theta t}} + 61.5, \widetilde{\mathrm{Re}_{\theta t}}\right] \\ F_{\text{length}} &= \min\left[\exp\left(7.168 - 0.01173\cdot \widetilde{\mathrm{Re}}_{\theta_t}\right) + 0.5, 300\right] \end{aligned} \tag{3.40}$$

The coupling between the γ-Re_θ model and the standard SST turbulence model is implemented by modification of the k equation (shown in eq. (3.32)). The equation is modified as follows[48]:

$$\frac{\partial}{\partial t}(\rho k) + \frac{\partial}{\partial x_i}\left(\rho k u_i\right) = \frac{\partial}{\partial x_j}\left(\Gamma_k \frac{\partial k}{\partial x_j}\right) + G_k^* - Y_k^* + S_k \tag{3.41}$$

$$G_k^* = \gamma_{eff}\cdot \widetilde{G_k} \tag{3.42}$$

$$Y_k^* = \min\left(\max\left(\gamma_{eff}, 0.1\right), 1.0\right) \cdot \widetilde{Y_k} \tag{3.43}$$

Where $\widetilde{G_k}$ and $\widetilde{Y_k}$ are, respectively, the original production and destruction term of turbulent kinetic energy shown in eq. (3.32).

The γ-Re$_\theta$ imposes new mesh requirements: that the mesh prism layer must have a first cell centroid y$^+$ value close to one in order to accurately represent the laminar and transitional boundary layers[48]. In Fig. 3.12 and 3.13, the dependency between the transition point and first cell centroid y$^+$ for a flat plate is shown and compared with experimental results. As shown in Fig. 3.12, if the y$^+$ is too high (y$^+$>5), the starting position of the transition is moved upstream towards the leading edge of the flat plate. In contrast, if the value of the first cell y$^+$ is less than 0.001, the transition location tends to move towards the trailing edge of the plate. As shown in section 3.3.1.1, the first cell centroid y$^+$ was chosen to have a value between 0.001 and 10, as this range of values had little effect on the solution[48]. Indeed, during the performed simulations, the value assumed by y+ varied in the range 0.005-2. An inflation layer expansion factor greater than 1.2 causes a noticeable upstream offset of the transition point, presumably due to non-physical pressure gradients between cells[48]. Therefore, as shown in section 3.3.1.1, the prism layer growth factor was set as equal to 1.1.

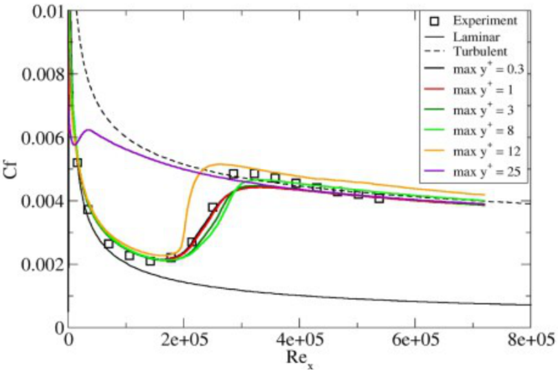


Fig. 3.12: Effect of an increasing first-cell centroid y$^+$ on a flat plate friction coefficient. Source: *ANSYS Inc.*, 2022[48].

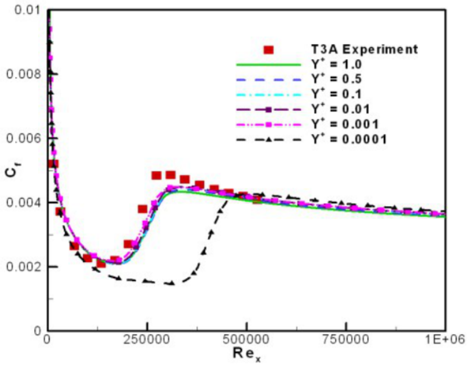


Fig. 3.13: Effect of a decreasing first-cell centroid y$^+$ on a flat plate friction coefficient. Source: *ANSYS Inc.*, 2022[48].

3.4.4 Near Wall Modelling and y$^+$ wall treatment

The presence of walls considerably impacts every flow, be it turbulent or laminar. The no-slip condition, which must be met at the wall and impose a null velocity on the body surface, significantly affects the velocity field. Moreover, the near wall region (particularly the buffer layer, see section 2.4) is where a significant amount of turbulent kinetic energy is produced and is one of the primary sources of vorticity in most flows[47,79]. In wall-bounded flows, the near-wall regions, where the flow fields have significant gradients, and the momentum and other scalar transports are most active successful predictions of the flow behaviour and the accuracy of numerical solution strongly depends on near-wall modelling and precise description of the flow in the region near the wall[40,47,79].

The near-wall method is usually modelled using two methods. The first approach uses wall functions and avoids modelling and resolving the turbulent boundary later innermost layers, the viscous sublayer and buffer layer, which are strongly impacted by viscosity[48,77]. To connect the viscosity-affected region between the wall and the fully turbulent zone, semi-empirical formulas. In the Near-Wall Modelling Method, the turbulence models are changed to allow the viscosity-affected region, which includes the viscous sublayer and the buffer layer, to be resolved with a mesh all the way to the wall[48,77]. This method requires a sufficient spatial resolution of the boundary layer mesh, requiring at least 20 cells to resolve the BL[48]. Moreover, the prism layer must be thicker than the boundary layer in order to prevent the border layer's growth from being constrained by the prism layer[48]. For more information on the model prism layer construction, see section 3.3.1.1.

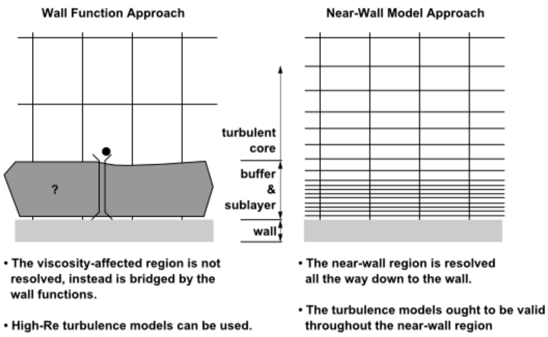

Wall Function Approach
Near-Wall Model Approach
turbulent core
buffer & sublayer
wall
?
• The viscosity-affected region is not resolved, instead is bridged by the wall functions.
• High-Re turbulence models can be used.
• The near-wall region is resolved all the way down to the wall.
• The turbulence models ought to be valid throughout the near-wall region

Fig. 3.14: Near-wall modelling of a wall-bounded flow and the two applicable approaches. Source: *ANSYS Inc.*, 2022[48].

3.4.4.1 The y^+-Insensitive Wall Treatment ω-Equation

The chosen k-ω turbulent model (see section 3.4.2) employs, by default, a y^+-insensitive wall treatment, which can be applied for both well-resolved prism layers with low y^+ values and coarse prism layers with high y_+ values. This is because the ω equation (eq. (3.34)) can be integrated all the way down to the wall through the viscous sublayer (in contrast with the ϵ equation on which the k-ϵ models family are based)[48]. However, to obtain a y^+ insensitive model, the viscous sublayer formulation and the logarithmic layer formulation have to be blended through a blending function to obtain a law-of-the-wall applicable to the entire near-wall region. Using a function proposed by Kader (1981)[88] this is accomplished by combining the linear (laminar) and logarithmic (turbulent) laws of the wall[48].

$$u^+ = e^{\Gamma} u^+_{lam} + e^{1/r} u^+_{\text{turb.}} \tag{3.44}$$

$$T^+ = \frac{\left(T_w - T_p\right) \rho c_p u^*}{\dot{q}} = e^{\Gamma_t} T^+_{lam} + e^{1/\Gamma_t} T^+_{\text{turb}} \tag{3.45}$$

$$\Gamma = -\frac{a\left(y^+\right)^4}{1 + by^+} \tag{3.46}$$

$$\Gamma_t = -\frac{a\left(Pry^+\right)^4}{1 + bPr^3 y^+} \tag{3.47}$$

Where a=0.001 and b=5.

3.4.5 Initial Conditions and Boundary Conditions

The following sections show the boundary and initial conditions applied to the simulated model. Additionally, the motivations behind the decisions made are highlighted. It should be noted how the boundary conditions described below depend on the values assumed by the free parameters of the parametric sweep (see section 3.2). Consequently, different BCs were applied for each simulated scenario.

3.4.5.1 Boundary Conditions: stationary walls and pressure far field

Quantity	Value
Gauge Pressure	p_∞ [Pa]
Mach Number	M_∞
Flow Direction	[1, 0]
Turbulent Intensity	0.5%
Turbulent Viscosity Ratio	10
Static Temperature	T_∞

Tab. 3.8: The Pressure Far-Field boundary conditions user-defined Values.

The external domain boundaries have been assigned the Pressure Far-Field Boundary Condition. The BC is defined by the variables shown in Tab. 3.8: Gauge pressure, Mach Number, Flow direction, Turbulent Intensity, Turbulent viscosity ratio and Static Temperature of the flow on the considered boundary.

Gauge Pressure The gauge pressure was set to p_∞ for every considered scenario. This is because ANSYS Fluent solves the NS equations by substituting the real pressure with the gauge pressure[48,77], defined as:

$$p_{\text{abs.}} = p_{\text{op.}} + p_{\text{gauge}} \tag{3.48}$$

Where p is the thermodynamic static pressure, and $p_{\text{op.}}$ is the user-defined operating pressure, which was set, for each and every simulated scenario, as 0. Consequently, the Gauge pressure was set as the freestream pressure p_∞ computed as described in section 3.4.5.3. This is done to reduce round-off errors during the solving process. Therefore, for all boundaries, as the operating pressure is equal to the freestream pressure, the gauge pressure was set to zero; for more information on the operating pressure, see sections 3.5 and 3.6.

Turbulent Intensity The turbulent intensity defined as:

$$TI = \frac{u'}{U} \tag{3.49}$$

where U is the mean velocity and u' is the root mean square of the turbulent velocity fluctuations (defined as u′=u-⟨u⟩=u-U), has been set to 0.5%. This value was considered acceptable and representative of the flow quality in a high-speed wind tunnel[40,89].

Turbulent Viscosity Ratio The turbulent viscosity ratio, defined as the ratio between the eddy viscosity and the molecular viscosity μ_t/μ could not be estimated. Additionally, as this parameter could have been used to fine-tune the heat conduction results, which could be argued to be a bad practice, it was set to the recommended value for external aerodynamics problems equal to 10[48,77].

Adiabatic Wall BC A wall boundary condition was assigned to the airfoil surface. The BC prescribes the no-slip condition (zero speed on the rigid stationary wall) and a zero heat flow condition. This is equivalent to wall adiabaticity. Consequently, the wall is expected to have a zero-tangent thermal boundary layer profile (see section 2.4).

3.4.5.2 Initial Conditions and Hybrid Initialization

The simulation's Initial conditions have been set thanks to the hybrid initialization method provided by ANSYS Fluent. It solves two Laplace equations to produce a pressure field that smoothly connects high and low-pressure values in the computational domain and a velocity field that conforms to complex domain geometries. The patching of all other variables, such as temperature and turbulence, is carried out using domain averaged values[48]. Compared to the standard, uniform values initialization, hybrid initialization aims to speed up convergence in the first iterations.

Velocity Field The velocity field is obtained by solving the Laplace equation for the velocity potential and applying the proper boundary conditions.

$$\nabla^2 \varphi = 0 \tag{3.50}$$

Where φ is the velocity potential, defined as

$$\vec{V} = \nabla \varphi \tag{3.51}$$

The velocity normal to the wall is set to zero in a potential flow. Consequently, on a wall:

$$\left.\frac{\partial \varphi}{\partial n}\right|_{\text{wall}} = 0 \tag{3.52}$$

On a far field boundary (the only inlet/outlet boundaries applied to the model) the user-specified free stream conditions are used to compute the velocity normal to the boundaries.

$$\left.\frac{\partial \varphi}{\partial n}\right|_{\text{inlet}} = V_{\perp} \tag{3.53}$$

Pressure Field To produce a uniform pressure field in the domain, a pressure Laplace equation is solved with the suitable boundary condition.

$$\nabla^2 p = 0 \tag{3.54}$$

On stationary walls, the pressure gradient normal to the surface is set to zero

$$\left.\frac{\partial p}{\partial n}\right|_{\text{wall}} = 0 \tag{3.55}$$

Additionally, on inlet boundaries (such as the Pressure Far-Field boundary condition) the applied pressure on the boundary is the gauge total pressure, namely the total pressure on the boundary minus the set operating pressure on the model.

$$p_{boundary} = p^0_{boundary} - p_{op}. \tag{3.56}$$

Temperature Field The temperature is initialized on the entire domain with a constant value equal to the average of the specified temperatures on the boundaries.

Turbulent Parameters Constant domain averaged values are used to initialize the turbulent parameters.

3.4.5.3 Freestream conditions parametrization

As was shown in section 2.2, the body forces on a rigid body are essentially governed by the body geometry and the freestream flow Mach number and Reynolds number. As previously mentioned, the Pressure Far-Field BC, which is specified by the freestream Mach number, pressure and temperature, was applied to the model inlet and outlet boundaries. As such, the proper pressure and temperature value to apply to the BC needed to be found for every considered combination of M and Re.

Starting with the definition of Mach number and Reynolds number:

$$M_\infty = \frac{V_\infty}{a}, \quad Re_\infty = \frac{\rho V_\infty L}{\mu} \tag{3.57}$$

and substituting the following expression is obtained.

$$Re_\infty = \frac{\rho(p,T) M_\infty c(T) L}{\mu} \tag{3.58}$$

Substituting the ideal gas law ($p = \rho RT$) and the speed of sound formula ($c = \sqrt{\gamma RT}$) it can be shown that:

$$Re_\infty = \frac{p M_\infty \sqrt{\gamma RT} L}{RT\mu(T)} = \frac{p M_\infty \sqrt{\gamma} L}{\sqrt{RT}\mu(T)} \tag{3.59}$$

Where the dynamic viscosity is a function of temperature according to Sutherland's Law. The previous expression can be solved either for pressure as a function of temperature or as temperature as a function of pressure. However, due to the Sutherland model for viscosity, solving for temperature needs an iterative method. Therefore, it was decided to solve for pressure as a function of temperature.

$$p = \frac{Re_\infty \mu(T) \sqrt{RT}}{M_\infty L \sqrt{\gamma}} \tag{3.60}$$

Note that the equation has 3 degrees of freedom. For this reason, it was decided to have a constant temperature of 243.15 K (or -30℃). Subsequently, the freestream dynamic viscosity was also fixed as a constant equal to μ_∞=$f(\mathrm{T}_\infty)$ according to Sutherland's Law. The pressure and temperature boundary conditions are shown in eq. (3.61).

$$\begin{cases} T_\infty = 293.15 \quad \mathrm{K} & (3.61.a) \\ p_\infty = \dfrac{Re_\infty \mu(T_\infty)\sqrt{RT}}{M_\infty L \sqrt{\gamma}} \quad \mathrm{Pa} & (3.61.b) \end{cases}$$

3.5 Solver options

The following sections describe the most relevant solver settings and highlight the reason such choices were made. Some of the applied solver settings are summarised in Tab. 3.9.

3.5.1 Timestep size and total physical time

Quantity	Value
Timestep	Fixed
Timestep size	0.0002 s
Number of timesteps	1500
Total flow time	0.3 s
Maximum inner iterations	10
Flux Scheme	AUSM+
Courant number	100
Multi-grid coarsening	5
Operating pressure	0

Tab. 3.9: Summary of the most relevant solver settings.

Timesteps were set to be of fixed size. The choice to adopt a fixed timestep rather than an adaptive one was made because doing so allowed the dataset's number of features to be reduced (see section 4.3). The simulation data, such as rake flow field values and the airfoil dimensionless coefficient values, are time-dependent. Therefore, each exported data item would need to include the relevant physical time if the timestep had been configured to be simulation-dependent: given how the data was encoded, this would have been equal to adding a data channel to the 2D image tensor of the exported data, which is more precisely equivalent to adding a feature column to the dataset. Section 4.3 can be read for further details on the dataset structure. As a result, a new channel and more trainable parameters would have been added to the neural network. This was not deemed preferable due to the small size of the dataset. Consequently, the fixed timestep size was set to 0.0002 s. This value resulted from multiple trials to obtain a timestep size that would allow a moderate number of maximum inner iterations (section 3.5.2) to be used and allow for an appropriate accuracy and temporal resolution.

The simulation total physical time was established both on a physical basis and by iteratively running multiple simulations to establish the minimum required runtime to resolve periodic unsteady flow features accurately and to obtain accurate unsteady values of the nondimensional coefficients C_d, C_l, C_m.

3.5.2 Flux schemes and convergence

As previously stated, a density-based implicit transient solver was used. The evaluation of the flows on the cell faces is carried out in ANSYS Fluent using two possible schemes: Roe-FDS or AUSM+. Given the presence of supersonic shocks on the airfoil's upper surface, the AUSM+ scheme was chosen. With a comparable computational cost[90], the AUSM+ scheme has several desirable properties: unlike the ROE-FDS scheme, it does not suffer from shock instabilities and carbuncle flux oscillations[91]. Furthermore, it preserves the positivity of scalar quantities (unlike the Roe-FDS scheme, which can lead to the calculation of negative densities)[48,90] and does not introduce artificial dissipation terms and is free from supersonic shock flux oscillations[48,90]. Furthermore, in the case of tetrahedral and triangular grids, it generates the most accurate results for supersonic shocks[92].

The inner iterations pseudo-timestep required to solve the algebraic system at each timestep is governed by the user-defined implicit transient solver Courant number[48]. It was iteratively set to reach a good convergence of the inner iterations in as few iterations as possible without compromising simulation stability[48].

3.6 Data acquisition

The dimensionless airfoil coefficients (C_l, C_d and C_m) data were exported for each timestep. Rakes, surfaces with a predetermined number of nodes evenly spaced between two predetermined endpoints[77], were used to sample the wake temperature, pressure, and velocity data. Rakes simulates the instrumentation used to measure experimental data in wind tunnel testing[93]. Fig. 3.15 displays the rakes, each rake's ID, and the position relative to the airfoil. Additionally, the number of nodes on each rake, their size, and the spacing between nodes are tabulated in Tab. 3.11. Instead, the physical quantities exported on each node are shown Tab. 3.11. It should be noted that the rakes' nodes have no connectivity. Therefore, there is no guarantee that a rake node coincides with a cell centroid[77]: when the rakes' node does not coincide with a cell centroid, the value on the node is

Exported training field variable	Exported control field variable
Static pressure [Pa]	Flow time [s]
Velocity magnitude [m/s]	Node XY coordinates [m]
Velocity Angle [deg]	Total pressure [Pa]
Static temperature [K]	

Tab. 3.10: Exported field variables on the rake nodes. In the left column are the variables later used to train the neural network. In the right column, the variables used as a control to confirm the accuracy of the exported data.

Rake ID	Rake Length [m]	N. of nodes	Node spacing [m]
1	2	201	0.01
2	3	301	0.01
3	2	201	0.01
4	3	301	0.01
5	4	401	0.01

Tab. 3.11: Rake ID's (shown in Fig. 3.15) and corresponding geometric properties.

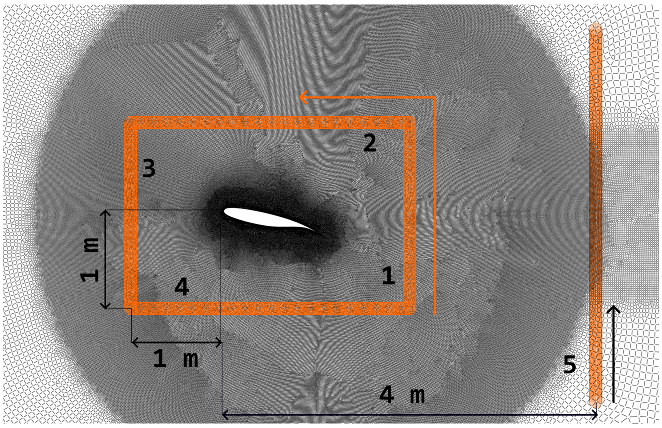


Fig. 3.15: Position of the rakes relative to the airfoil. The corresponding identification number is shown next to each rake. The orange arrows indicate the direction of rake data acquisition and storage, while the size and number of nodes on each rake are shown in the table.

computed by interpolation with the actual value of the surrounding cell centroids. Node values interpolation involves two steps: first, the initialization sets the values to the weighted average of adjacent cell values, with weights being inverses of the cell volumes neighbouring the nodes. Then, at boundaries, the initially computed node values are replaced by the straightforward average of the boundary face values[48]. The flow field values exported are shown in Tab. 3.10; note that only some were used to train the neural network.

3.6.1 Fluid flow data sets

The collected data were used to compile 3 different datasets. Each one shares the airfoil performance tensor (for the derivation and numerical procedure, see section 4.3). The datasets are differentiated by the data collected on the rakes. Each of the proposed neural network architectures was trained with each of the 3 datasets to analyze the different learning capacities of the networks as the completeness of the near-field information (see sections 3.6.1.1 and 3.6.1.2) and the quality and spatial resolution of the wake data varied (see sections 3.6.1.2 and 3.6.1.3).

3.6.1.1 Box rake data set

The data from dataset 1 were continuously collected on rakes 1-4, as shown in Fig. 3.15. It should be noted that as a result, the last data point collected on rake 4 and the first data point collected on rake 1 relate to physically adjacent nodes. This dataset contains the most information about the near field. Consequently, higher model performance is expected from neural networks trained with this dataset.

3.6.1.2 Wake rake data set

The data from dataset 2 was collected along the nodes of rake 1. As such, the dataset only refers to wake data.

3.6.1.3 Far wake rake data set

The data from dataset 3 was collected along the nodes of rake 5. As shown in Fig. 3.15, this rake is the furthest from the trailing edge of the wing, and most of the rake surface falls outside the wake of the airfoil. As a result, neural networks trained with this data set are expected to have lower accuracy.

3.7 Model validation

Parameter	Value
Angle of attack (AoA)	3.0°
Reynolds number (Re)	$1.5 \cdot 10^7$
Mach number (M)	0.725

Tab. 3.12: Value of the parameter sweep free parameters employed in the model validation.

The model described in the previous sections was validated by comparison with the experimental data provided by Jenkins (1989)[94] and Bartels and Edwards (1997)[95]. As previously pointed out, the maximum accuracy of the similarities was different from the goals of this work due to the time and computational cost limitations imposed by the number of simulations to be carried out to construct the datasets. Given the preliminary nature of this work, the primary aim was to create a dataset of data with obvious physical sense, even if not of absolute accuracy. This choice was partially imposed by the fact that few experimental data on dimensionless airfoil coefficients in the transonic regime are accessible in the literature. In addition, almost no data regarding coefficients in the post-stall field at high AoAs can be found in the literature; consequently, a validation for high angles of attack was not carried out, and the model was only validated for low AoAs.

Model validation was performed by comparing the pressure coefficient profile on the airfoil for a physically stationary case at low AoA. Experimental data were obtained from Jenkins (1989)[94] and Bartels and Edwards (1997)[95]. The simulations were conducted according to the same methodology reported in the previous sections, and the values of the freestream parameters are shown in Tab. 3.12. As can be seen in Fig. 3.16, the pressure coefficient values collected through CFD simulations showed a good correspondence with the experimental results collected in the wind tunnel. It can be seen that the CFD simulations predict a higher back pressure near the leading edge than the wind tunnel tests found. This was expected, as it is well known that RANS simulations of 2D airfoils tend to overestimate the lift and underestimate the drag of airfoils.

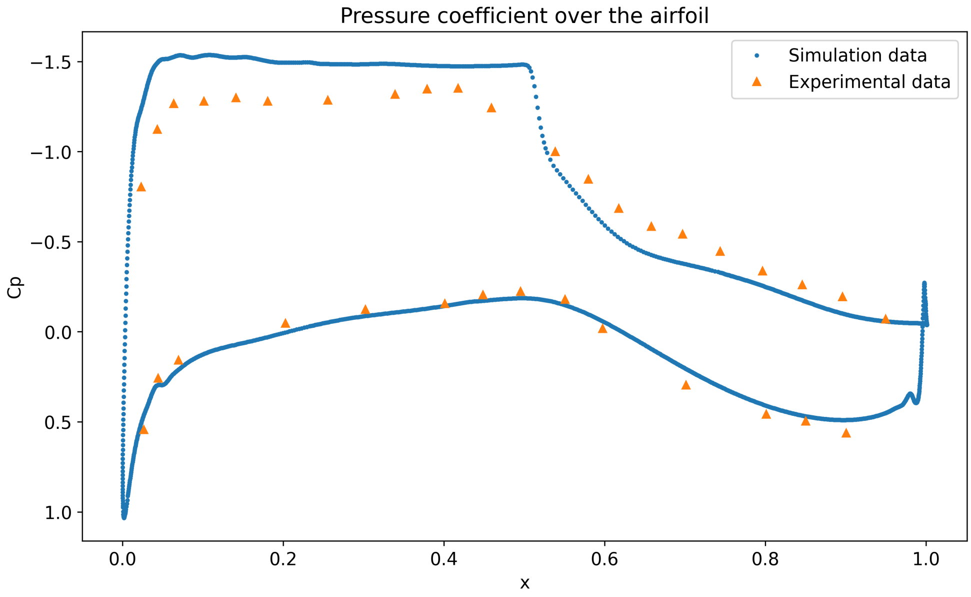


Fig. 3.16: Pressure coefficient over the airfoil for AoA=3°, Re = $1.5{\cdot}10^7$, M=0.725. CFD results and experimental results are overlayed.

Chapter

Neural Networks architecture & training

4

This chapter outlines the methodology for processing simulation data and designing and implementing the deep-learning models. The architectures of the proposed neural networks are explored, their main components are listed, and the design choices are explained based on the literature and similar works.

The proposed networks were generated on TensorFlow[96], an open-source library for artificial intelligence developed by Google, and the multi-branch, more complex architectures were generated thanks to the TensorFlow Functional API[58].

4.1 Problem description

The proposed neural networks aim to predict the aerodynamic performance of an airfoil in a compressible field as the angle of attack, Reynolds number, and Mach number vary (from now on, these quantities will be referred to as freestream parameters). As expanded in Section 232, the time series of the airfoil aerodynamic performance (i.e., the time-series of the 3 dimensionless coefficients C_l,C_d,C_m) are expressed as the amplitudes and frequencies of the 3 principal harmonics of the time series of the 3 dimensionless coefficients C_l,C_d,C_m

This is achieved by using as input the scalar parameters characterizing the free stream (i.e. AoA, Re, and M) and near-field temperature, pressure, and flow velocity wake data collected during every time-step of the CFD simulations on an envelope in proximity to the airfoil. The aim is to simulate a hypothetical experimental wind tunnel setup consisting of a rake positioned in the wake and near-field of the airfoil.

The networks then search for a relationship

$$F(\alpha, Re, M, \boldsymbol{T}, \boldsymbol{p}, \boldsymbol{V_{mag.}}, \boldsymbol{V_{ang.}}) \tag{4.1}$$

Since the proposed neural networks have a branched multi-input architecture in which scalars (the freestream parameters) and spatiotemporal matrix data (the rake data) are used as inputs, the function sought by the proposed neural networks can be formalized as:

$$F : \left(\mathbb{R}^{l} \cup \mathbb{R}^{m\times n\times o}\right) \longrightarrow \mathbb{R}^{p\times q\times r} \tag{4.2}$$

Where:

l is the number of freestream parameters α, Re, M. Its value is 3.

m is the number of rake nodes on which the near-field data has been collected on.

n is the number of time-steps in which the rake data has been collected. Its value is 750, i.e., the number of time-steps of the simulations actually used to calculate the aerodynamic performance of the airfoil.

o are the physical quantities whose values have been collected in the near field (T, p, $\mathrm{V_{mag.}}$, $\mathrm{V_{ang.}}$). Its value is 4.

p is the number of major harmonics obtained by the Fourier Transform of the time series of dimensionless coefficients. As previously stated, it is equal to 3.

q is the data of the three major harmonics. Its value is 2, as only the frequency and magnitude for each harmonic have been calculated.

r is the number of nondimensional coefficients (i.e. $\mathrm{C_l}$, $\mathrm{C_d}$, $\mathrm{C_m}$) that have been collected. Its value is 3.

The value of m, the number of rake nodes on which the near-field data has been collected, differs for each of the three datasets considered (see section 3.6). The number of nodes used for each dataset is given in Tab. 3.11 and 4.1. Consequently, three different networks were produced for each of the proposed models, which differ in the physical meaning of the datasets (full near-field envelope, near wake data and far wake data) and the dimension of the inputs.

A fourth neural network (NN0) consisting of a simple FFNN was also trained to compare the proposed models with a network of lower complexity. The proposed FFNN has a simple single-input architecture in which scalars (the freestream parameters) are used as inputs, and the principal harmonics are used as network outputs. The function sought by the proposed neural networks can be formalized as:

$$G : \mathbb{R}^{l} \longrightarrow \mathbb{R}^{p\times q\times r} \tag{4.3}$$

Dataset Name	# of timesteps	# of rake nodes
Parameters & Harmonics (Dataset 0)	N.A.	N.A.
Box Rake (Dataset 1)	750	1000
Wake Rake (Dataset 2)	750	201
Far Wake Rake (Dataset 3)	750	401

Tab. 4.1: Names, number of time-steps and number of nodes of the rakes on which data were collected for each dataset

4.2 Dataset structure and encoding

The simulation data was encoded in tensors to be used for training and testing the neural network. As previously explained, the architectures considered are hybrid convolutional-fully connected architectures. Therefore, the rake data has been encoded as a 2D image with multiple stacked channels.

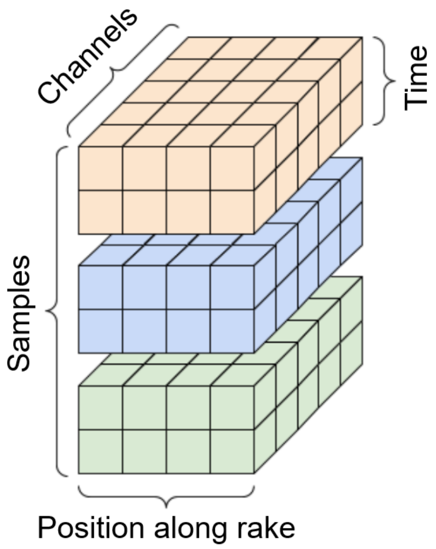


Fig. 4.1: Three-dimensional representation of a rank 4 tensor. The tensor shape represents how the simulation data were stored. Source: *Google LLC*, 2023[58]. Modified by the author.

Thus, for each sample, the near-field data has dimension $\mathbb{R}^{m \times n \times c}$. Where $m \times n$ is the size of the image. **n** is the number of time-steps used for training and thus encodes temporal information; **m** is the number of nodes along which rake data were collected and thus encodes spatial information. Each of these 2D images encodes the data on all rake nodes and during all time-steps for one of the physical quantities considered (e.g., the static pressure p). **c** is the number of channels, corresponding to the number of physical quantities collected from the rakes; thus, each of the 2D images containing the near-field data for the physical quantities considered (T, p, $V_{mag.}$, $V_{ang.}$) is thus stacked along dimension c. The single sample near-field data is thus encoded as a 3D tensor. The entire dataset dimension is $\mathbb{R}^{S \times m \times n \times c}$, as the 3D tensors encoding the individual samples of near-field data are stacked along the S axis, which indicates the number of samples. A visualization of the data structure that was just stated is depicted in Fig. 4.1.

In contrast, the freestream parameters have no spatial or temporal component, as the freestream parameters are kept constant during the simulations. The 3 parameters AoA, Re, and M were thus stored, for each dataset sample, as an array in the space $\mathbb{R}^{p}$, where p=3. The entire dataset freestream parameter dimension is thus $\mathbb{R}^{S\times p}$, as the 1D arrays encoding the individual samples of near-field data are stacked along the S axis, which indicates the number of samples.

The neural network output consists of the frequency and amplitude of the 3 primary harmonics of the time series of the aerodynamic performance coefficients. The output for each individual sample and individual coefficient has dimension $\mathbb{R}^{f\times g}$, where f corresponds to the number of harmonics considered, and g corresponds to the calculated quantities (i.e., frequency and amplitude). Thus, for each individual sample, the output is a tensor in the space $\mathbb{R}^{f\times g\times h}$, where h is equal to the number of dimensionless coefficients considered (i.e., 3). The entire dataset output dimension is thus $\mathbb{R}^{S\times f\times g\times h}$, as the 3D arrays encoding the individual samples are stacked along the S axis, which indicates the number of samples in the dataset.

A fourth dataset was compiled (named (Dataset 0) Parameters & Harmonics) consisting only of the freestream parameters as input and the principal harmonics of the aerodynamic coefficients as output. This dataset was used to train and perform hyperparameter optimization on a simple FFNN (see Section 2324).

4.3 Data preparation & Storage

The data obtained from the 343 simulations carried out (see section 3.2 and Tab. 3.1) were used to compile three different datasets. The 343 simulations were obtained by varying the three freestream parameters (AoA, Re, and M). Each of the 3 freestream parameters was assigned 7 equispaced values, and a simulation was carried out for each combination of parameter values considered. The values assigned to the parameters during the sweep are given in Tab. 3.1. The data obtained from the simulations are:

- freestream parameters, which govern the parameter sweep performed.
- the near-field temperature, static pressure, velocity magnitude, and velocity angle time-series data. Data were collected on rake nodes, surfaces that approximate the experimental setup for measuring data. The location of the rakes and the number of nodes are shown in Fig. 3.15 and Tab. 3.11.

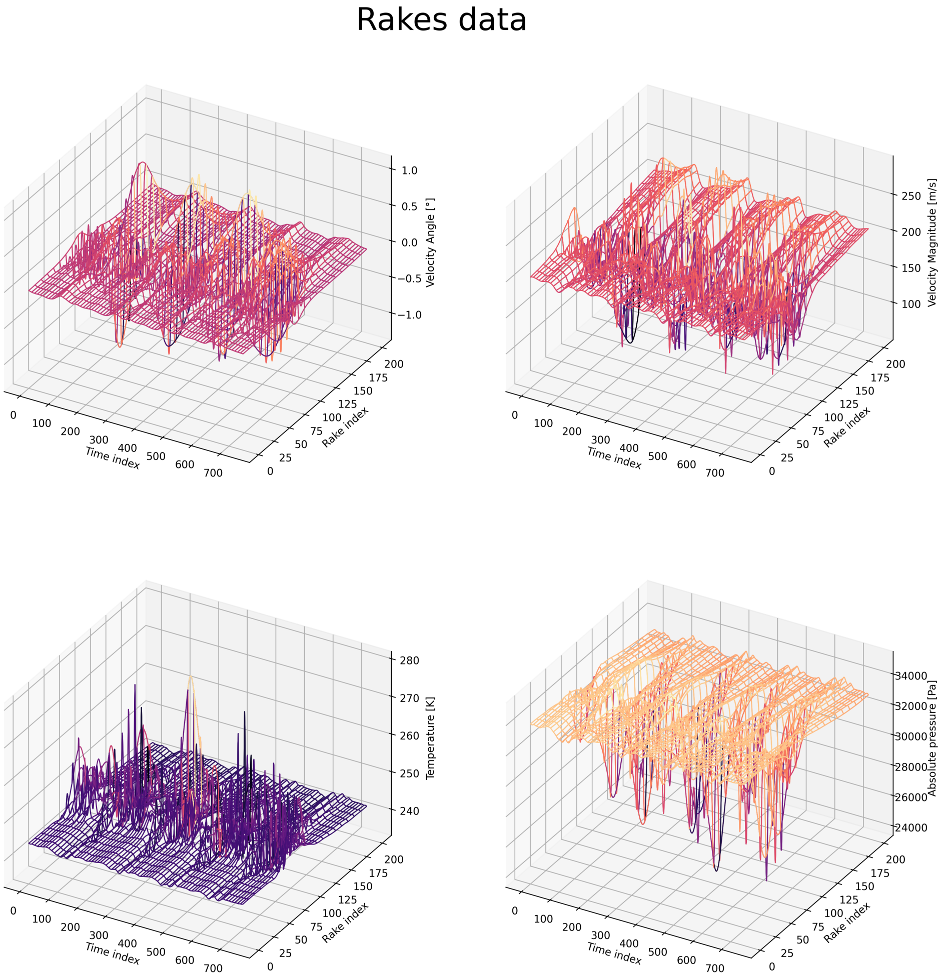


Fig. 4.2: Visualization of rakes' data. The data shown belongs to the database 2 and refers to the scenario characterized by freestream parameters AoA=9°, Re=$6.25{\cdot}10^6$, M=0.65.

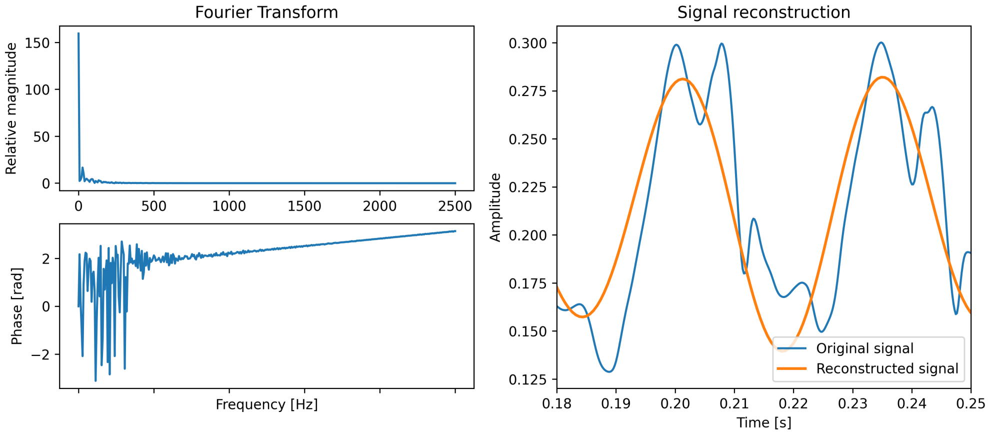


Fig. 4.3: Fourier transform of the C_d time series (for AoA=9°, Re=$6.25{\cdot}10^6$, M=0.65), signal analysis, and approximate reconstruction using the DC component and the two dominant harmonics.

- time-series of the dimensionless aerodynamic coefficients, i.e., the value of the coefficients C_l, C_d, C_m over time.

Freestream parameters and rake data were used as inputs to the neural networks. The encoding of the data is discussed in detail in section 4.2. The time series of the aerodynamic performance coefficients C_l, C_d, and C_m were not directly used as outputs of the proposed networks. However, a Fourier transform was performed on the time-series data to obtain, for each of the 3 dimensionless coefficients, the amplitude and frequency of the 3 primary harmonics of the time-series. The data thus obtained was used as the output of the proposed neural networks. More detailed information on the procedure is given in section 4.3.2.

The resulting data were used to compile 3 datasets, which differ in the location and number of rake nodes on which the near-field temperature, pressure, and velocity data were collected. It is good to note that all 3 datasets share the freestream parameters as inputs to the networks and the principal harmonics of the aerodynamic coefficient time series as outputs. The datasets differ only in the near-field data's acquisition location and number of acquisition nodes. The 3 compiled datasets are thus:

Parameters & Harmonics (Dataset 0) The dataset consist only of the freestream parameters as input and the principal harmonics of the aerodynamic coefficients as output. This dataset was used to train and perform hyperparameter optimization on a simple FFNN (see Section 2324).

Box rake data set (Dataset 1) The data from dataset 1 were continuously collected on rakes 1-4, as shown in Fig. 3.15. It should be noted that as a result, the last data point collected on rake 4 and the first data point collected on rake 1 relate to physically adjacent nodes. This dataset contains the most information about the near field. Consequently, higher model performance is expected from neural networks trained with this dataset.

Wake rake data set (Dataset 2) The data belonging to dataset 2 was collected along the nodes of rake 1. As such, the dataset only refers to wake data.

Far wake rake data set (Dataset 3) The data belonging to dataset 3 was collected along the nodes of rake 5. As shown in Fig. 3.15, this rake is the furthest from the

Dataset Name	Freestream parameters input	Rakes data input	Principal harmonics of the coefficient time series
	[samples×n. of parameters]	[samples×timesteps×# of nodes×channels]	[samples×# of harmonics×frequency and amplitude×# of coeffs.]
Box rake data set (Dataset 1)	343×3	343×1000×1000×3	343×3×2×3
Wake rake data set (Dataset 2)	343×3	343×1000×201×3	343×3×2×3
Far wake rake data set (Dataset 3)	343×3	343×1000×401×3	343×3×2×3

Tab. 4.2: Names, input and output sizes of the databases used in neural network training.

trailing edge of the wing, and most of the rake surface falls outside the wake of the airfoil. As a result, neural networks trained with this data set are expected to have lower accuracy.

The input and output sizes of the 3 datasets are shown in Tab. 4.2. Thus, 3 variants (one per dataset) were produced for each of the proposed architectures, differing in the size of the rake data input and the size of the upsampled freestream data.

The processes of data clean-up, normalization of the input data, and Fourier transform are discussed in the following subsections.

4.3.1 Data clean-up

The results of the first time steps are the consequence of a transient between the initialized field and the real flow field of the simulated system. The data of the first 700 time-steps of the simulation were considered non-physical and not fit to train the neural network as they were overly dependent on the ICs of the simulations. Consequently, for each dataset, the data from the first 700 time-steps were not taken into account, and the dataset is comprised only of the last 750 time-steps.

4.3.2 Fast Fourier Transform of the nondimensional coefficients time-series

As mentioned previously, the airfoil performance prediction networks (see section 4.11) use the scalar parameters of the freestream and near-field data to predict the airfoil performances, which are encoded as the amplitudes and frequencies of the 3 principal harmonics of the time series of the nondimensional airfoil coefficients (i.e. $\mathrm{C_d}$, $\mathrm{C_l}$, $\mathrm{C_m}$).

The nondimensional coefficient time series were processed by applying a Fourier transform to obtain the dominant harmonics of the time series. The dominant harmonics were then identified, and the original signal was reconstructed by extrapolating the DC component of the signal (i.e. the 0 Hz harmonic) and the magnitude and phase of the successive two dominant harmonics. A visualization of the process of time series transformation and subsequent signal reconstruction is shown in Fig. 4.3. The reconstructed signals were then encoded, for each sample, as a 3×2×3 tensor, where the first axis refers to the 3 dominant harmonics, the second axis refers to the magnitude and frequency of the dominant harmonics, and the third axis refers to the various nondimensional coefficients (in order,C_d, C_l, C_m).

4.3.3 Normalization

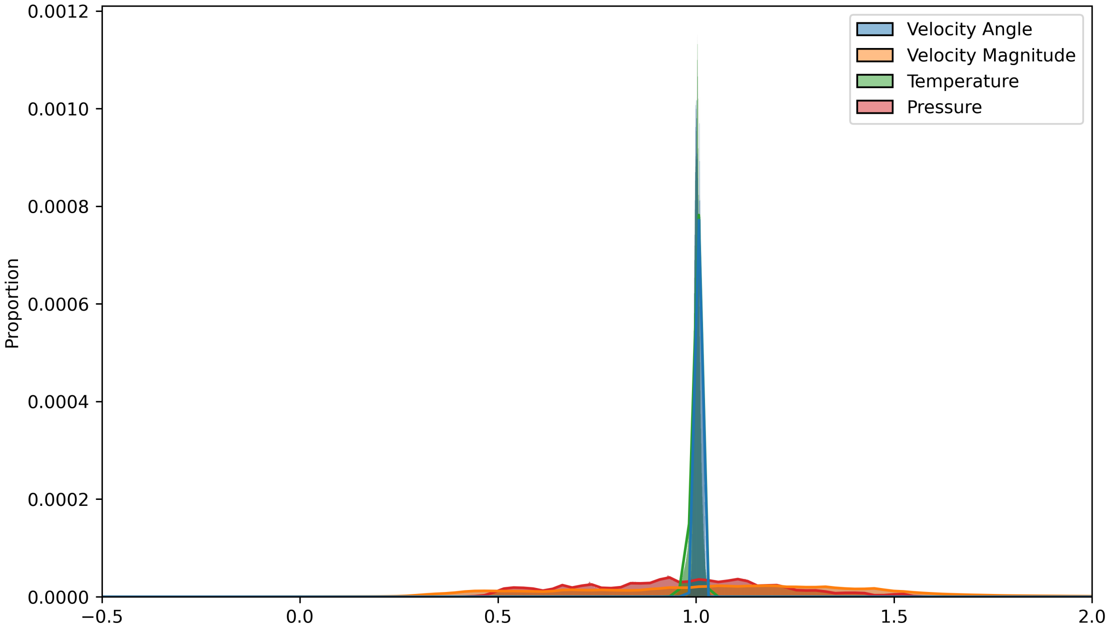


Fig. 4.4: Distribution of the Rakes data after being normalized using the values given in Tab. 4.3.

Data normalization aims to obtain data with a similar scale, i.e., a comparable numerical value. The importance of normalization in obtaining high-quality and robust models has long been recognised[97].

The scalar parameters characterizing the free stream (i.e. AoA, Re, M) have been normalized by scaling over a fixed range as shown in eq. (4.4).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.4}$$

where x is the normalized quantity, x_{min} is the minimum value assumed by that quantity and x_{max} is the maximum value assumed. Fixed range normalization could be applied because the values and distribution assumed by the scalars characterizing the freestream were known *a priori*, having been set and varied during the parameter sweep (see section 3.2). Therefore, the normalization values applied are the minimum and maximum parameter sweep values given in Tab. 3.1.

The near-field flow data acquired on the rakes did not appear to follow a normal or otherwise simple distribution along each of their axes. For this reason, it was decided not to standardize the data and scale the rake data over a fixed range according to the physical quantity represented. Each of the physical quantities (V_{angle}, V_{mag}, T, p) was normalized according to the formula in eq. (4.4): the normalization values applied are shown in Tab. 4.3. It should be noted that the applied values are known a priori and are derived from a physical knowledge of the simulated system, as they refer to the maximum and minimum values assumed in the freestream during the parameter sweep by the relevant physical quantities (see section 3.4.5.3). The fixed range normalization could be applied equally to all datasets and splits, as the a priori knowledge would not cause any information leakage from the training dataset to the validation and test datasets.

The neural network output was normalized by z-scaling, which usually involves transforming each feature in the dataset to have a mean of 0 and a standard deviation of 1. For the present work, z-scaling was done to ensure a mean of 1 and a standard deviation of 1. This choice was imposed by the metrics chosen (see the dedicated section 2424) that measure relative error. An output mean of 0 would have led to a floating-point exception because of the division by 0. A z-scaling of 1 prevents this. This is achieved by applying the following formula to each feature:

$$z = \frac{x - \mu}{\sigma} + 1 \tag{4.5}$$

Where x is the feature data, z is the normalized feature data, μ is the feature mean, and σ is the feature standard deviation. It should be noted that the mean and standard deviation data used for normalization of the output are referenced to the training datasets to avoid information leakage from the training datasets to the validation and test datasets.

4.3.4 Database shuffling and splitting

The data was shuffled before using the datasets to train the neural networks. This was done to prevent the model from being influenced by the order of the data. Additionally, the training data was shuffled between epochs, to prevent the models from memorizing the order of the training data.

Physical quantity	**Channel**	$\mathbf{x_{max}}$	$\mathbf{x_{min}}$
Velocity angle [rad]	1	$\frac{\pi}{2}$	0
Velocity magnitude [m/s]	2	$\mathrm{M}_{\infty,\mathrm{max}}\cdot\mathrm{c}(\mathrm{T}_\infty)$	$\mathrm{M}_{\infty,\mathrm{min}}\cdot\mathrm{c}(\mathrm{T}_\infty)$
Temperature [K]	3	T_∞	0
Pressure [Pa]	4	$\frac{Re_{\infty,max}\mu(T_\infty)\sqrt{RT}}{M_{\infty,min}L\sqrt{\gamma}}$	$\frac{Re_{\infty,min}\mu(T_\infty)\sqrt{RT}}{M_{\infty,max}L\sqrt{\gamma}}$

Tab. 4.3: Normalization values of the rakes data. for more information on the values chosen, see section section 3.4.5.3.

The datasets have been split into three subsets, resulting in a training set to train the networks, a validation set to verify the models' effectiveness, and a test set. Due to the small dataset, the validation and test split used are respectively 0.2 and 0.1: that is, 20% of the original datasets were reserved for model validation, and an additional 10% were reserved for model testing. After the split, it was manually checked that the validation and test datasets were representative of the original datasets and did not have different characteristics from the original dataset.

4.4 Activation Functions

The effect of using different activation functions was investigated. The functions considered are Rectified Linear Unit (ReLU), GELU, and Swish. All 3 activation functions are plotted in Fig. 4.5. Their formulas are shown below:

$$\mathrm{RELU}(x) = max(0, x) \tag{4.6}$$

$$\mathrm{GELU}(x) = \frac{1}{2}x\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \tag{4.7}$$

$$\mathrm{SWISH}(x) = x\,\mathrm{sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}} \tag{4.8}$$

Database Split	**# of samples**
Total	343
Training	240
Validation	68
Test	35

Tab. 4.4: The number of samples in each database split. The same split was applied to all 3 datasets used for training the networks and described in section 4.2.

The three activation functions were set as hyperparameters of the middle layers of the hypermodels to guarantee the use of the activation function that guarantees the best model per architecture. For each hypermodel, the ReLU activation was set as the default. Similarly, the base models evaluated prior to the optimization of the hyperparameters were constructed using the ReLU activation. Since the problem addressed is regressive, the output layers of the proposed models have a linear activation function[61].

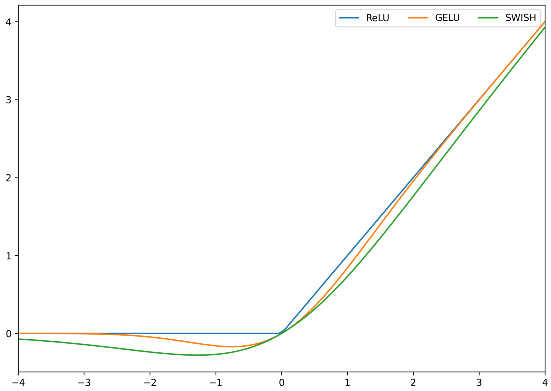


Fig. 4.5: Plot of the ReLU, GELU ans Swish activation functions in the range [-4, 4].

At the time of writing, no other work is known in which models employing GELU or Swish activation functions have been used to predict the aerodynamic performance of wings or airfoils. This was not considered a problem as Swish and GELU were considered valid to be a drop-in replacement for ReLU[63].

4.5 Loss function & performance evaluation metric

For each proposed architecture, the loss function implemented is the Root Mean Squared Relative Error (RMSRE). Initially, the use of Mean Square Error (MSE), a common choice for regressive networks, was considered; however, MSE was discarded because preliminary tests showed that the loss function RMSRE produced networks with better generalization capability, both in terms of RMSRE metric and MSE metric. The formula for calculating the RMSRE is shown in eq. (4.9).

$$\boldsymbol{RMSRE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{(\hat{y}_i - y_i)^2}{\hat{y}_i{}^2}} \tag{4.9}$$

As a relative function, RMSRE measures the relative error between the ground truth and the output of the neural network and exactly as MSE characteristic of the RMSRE is that it strongly penalizes outliers.

MSE was kept as a metric to ensure multiple ways of evaluating the performance of neural networks were available. The formula for calculating the MSE is shown in eq. (4.10).

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{4.10}$$

4.6 Regularization

Given the small dataset, the use of regularisation techniques to reduce the risk of overfitting was considered necessary[26,27,98], even tough CNNs are particularly resilient to overfitting[26]. This was achieved by employing Dropout Regularization and L_2 regularization. The combination of those regularization techniques is not uncommon and was initially proposed by Srivastava et al. (2014)[56] in the work which introduced the Dropout method. The working of both regularization techniques is discussed in section 2.5.2

For every model proposed, the actual value of the dropout rate and regularization rates value have been determined by setting them as hyperparameters and performing hyperparameter optimization (see section 4.12).

4.7 Custom cylindrical padding

Similar works usually employ zero-padding, where the tensor is padded with zeros[9] Although this preserves the tensor dimensionality, the operation is obviously non-physical. Furthermore, as reported in section 3.6, Dataset 1, which collects data from rakes 1, 2, 3, and 4, is cylindrical on the space axis. i.e., the data at the beginning and end of the space axis are referenced to adjacent points in the computational domain. In order to ensure a complete convolution of the edge pixels and to introduce padding with physical reasoning behind it, a custom padding layer was implemented in TensorFlow. An example of the custom padding layer implemented in TensorFlow is shown in eq. (4.11). The padding is here applied to a 2D slice of a 4D tensor. As can be seen, zero padding is applied along the temporal axis, while periodic padding is applied along the spatial axis.

$$\begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 & 26 \end{bmatrix} \xRightarrow[\text{Cylindrical Padding}]{} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 6 & 2 & 3 & 4 & 5 & 6 & 2 & 3 \\ 10 & 11 & 7 & 8 & 9 & 10 & 11 & 7 & 8 \\ 15 & 16 & 12 & 13 & 14 & 15 & 16 & 12 & 13 \\ 20 & 21 & 17 & 18 & 19 & 20 & 21 & 17 & 18 \\ 25 & 26 & 22 & 23 & 24 & 25 & 26 & 22 & 23 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.11}$$

It should be noted that although dataset 1 was the only periodic dataset along the spatial axis, cylindrical padding was used for each dataset. This is because, in addition to being considered closer to the actual physical conditions of the system, cylindrical (or periodic) padding leads to comparable or better results when applied to fluid-dynamic data, as demonstrated by Morimoto et al. (2021)[9]. Moreover, the padding algorithm follows the built-in implementation within TensorFlow Conv2D[58] convolutional layer. This is because, although not explicitly explained in the documentation, the padding applied by layers belonging to the Conv class differs in implementation and behaviour from the padding applied by the ZeroPadding2D[58] layer.

4.8 Pooling

The pooling operations considered in this work during the HPO process are max pooling and average pooling.

In addition, the proposed networks use Global Average Pooling (GAP) as an alternative to flattening at the network's end for dimensionality reduction and feature extraction. Global Average Pooling performs spatial pooling across the entire feature map of each convolutional kernel and allow for a reduction in the model parameters.

4.9 Scalar values input position

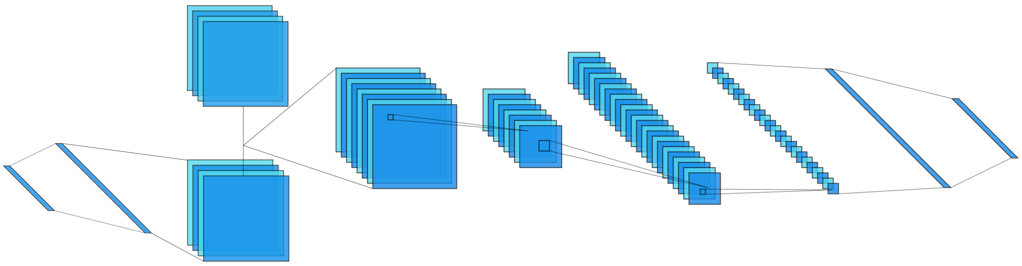


Fig. 4.6: The basic CNN shown in the figure employs the freestream data upsampling and reshaping used in all the proposed networks. A fully connected layer upsamples the freestream parameters array, and the output is upsampled and reshaped by a deconvolutional layer. The resulting tensor is concatenated with the near-field data tensor. The resulting 7-channel (4 channels encoding the near-field data and 3 channels encoding the freestream parameters upsampled data) is used as the input of the rest of the network.

As noted in section section 2.2, the drag and lift produced by a body immersed in a uniform freestream are governed by the characteristics of the freestream and the geometry of the body, i.e. shape and angle of attack.As such, in literature on predicting airfoil aerodynamic performance, freestream parameters are commonly used as additional inputs to neural networks[7,9–11]. Generally, for convolutional networks, the scalar inputs are added to the network in the last layers after flattening the output of the convolutional layers. However, as noted by Morimoto et al. (2021)[9], the predictive ability of CNNs for estimating aerodynamic performance can be improved, particularly for small datasets, by introducing the additional scalar inputs (AoA, Re, M) in the upstream layers. This was done by upsampling the scalar inputs via ConvTranspose2D[58] layers and concatenating the resulting matrix to the input tensor of the neural networks. The process is shown and expanded upon in Fig. 4.6. The approach of introducing scalar inputs into CNNs through upsampling is not unprecedented, and there are works in the literature that apply the same technique to other tasks[99].

4.10 Convolutional layers filters' size

Convolutional networks generally use square kernels (as the inputs to the networks are generally square tensors). However, preliminary tests on incomplete datasets highlighted that all proposed baseline models performed better with non-square kernels. This can be explained by the fact that the tensors' numerical data encodes data with different physical meanings on the first two axes (the first axis refers to the time-series, the second axis to the spatial location of the data; see section 4.2) with two different scales (the rakes data are collected over several metres, while the data are collected over a flow time of 0.15 s). For this reason, the kernels' height and width were set as hyperparameters of the hypermodels, and the use of a non-square kernel of size [3, 5] was initially considered for the baseline models but it was discarded in favour of a kernel size [3, 3] to reduce the number of model parameters. A default lower-size kernel was preferred as it was assumed that this could lead to a reduction in the number of trainable parameters and, consequently, facilitate the training process

4.11 Proposed architectures

As previously mentioned, multiple CNN-based neural network architectures were considered.

4.11.1 FeedForward Neural Network (FFNN)

The Feed-Forward architecture considered is the simplest in this paper. It consists of a sequence of fully connected layers followed by a single Dropout layer. It uses the freestream parameters as input and the principal harmonics of the aerodynamic coefficients as output. It is used as a benchmark to test the performance of convolutional networks against a simpler network that lacks the additional information given by the near field flow data obtained on the rakes.

4.11.2 Base CNN

The proposed basic CNN architecture (conceptually shown in Fig. 4.6) is the simplest convolutional network considered in this work. Comprising alternating convolutional layers and average pooling layers, each convolutional layer employs small filter sizes (5×5)to capture local features, followed by average pooling layers to downsample and extract the most relevant information, utilizing rectified linear unit (ReLU) activation functions throughout the network introduces nonlinearity and aids in feature extraction. The multi-branch input comprises fully connected layers and a Transposed Convolution layer to upsample the freestream input, which is then concatenated to the rake data (see section 4.9). The final layers consist of fully connected layers, a linear activation for the regression task, and a reshape layer to obtain the output tensor with the desired shape.

This basic CNN architecture is intended to learn the essential input data patterns while keeping the model's complexity low. It was used as a benchmark against which successive, more complex architectures were evaluated. The proposed architecture is shown in Fig. 4.8.

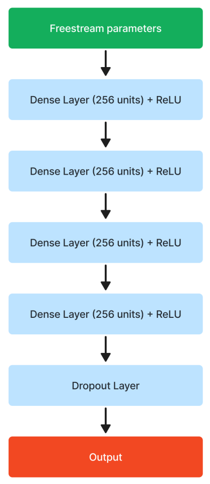

Freestream parameters
Dense Layer (256 units) + ReLU
Dense Layer (256 units) + ReLU
Dense Layer (256 units) + ReLU
Dense Layer (256 units) + ReLU
Dropout Layer
Output

Fig. 4.7: Flowchart of the proposed FeedForward Neural Network (FFNN) (NN0) architecture.

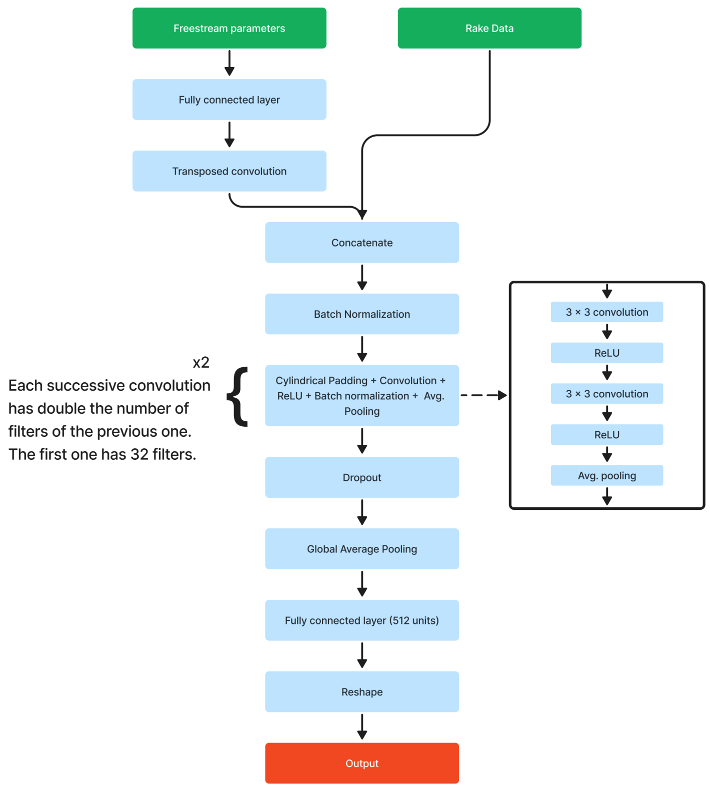
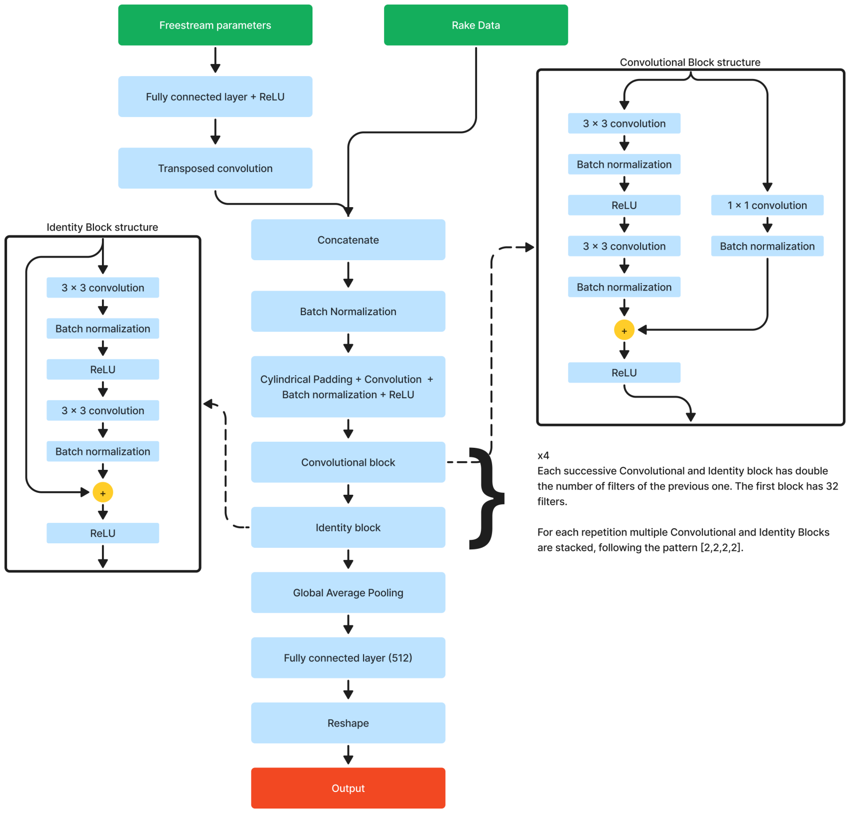

Freestream parameters
Rake Data
Fully connected layer
Transposed convolution
Concatenate
Batch Normalization
x2
Each successive convolution has double the number of filters of the previous one. The first one has 32 filters.
Cylindrical Padding + Convolution + ReLU + Batch normalization + Avg. Pooling
3 × 3 convolution
ReLU
3 × 3 convolution
ReLU
Avg. pooling
Dropout
Global Average Pooling
Fully connected layer (512 units)
Reshape
Output

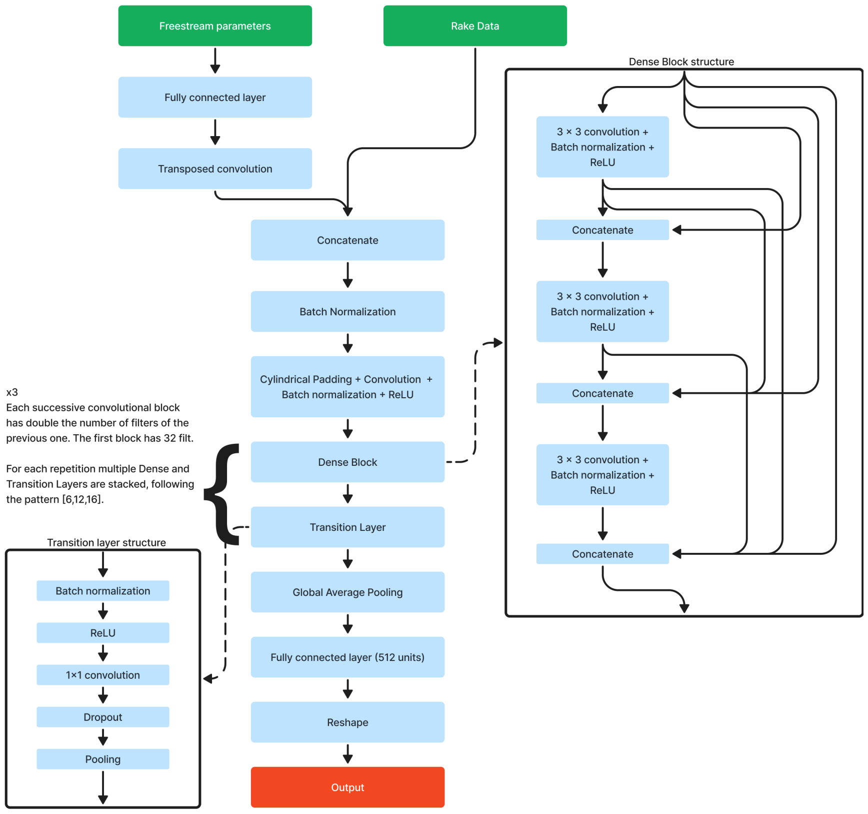
Fig. 4.8: Flowchart of the proposed Basic-CNN (NN1) architecture.



x4
Each successive Convolutional and Identity block has double the number of filters of the previous one. The first block has 32 filters.

For each repetition multiple Convolutional and Identity Blocks are stacked, following the pattern [2,2,2,2].

Fig. 4.9: Flowchart of the proposed ResNet-like (NN2) architecture.



x3
Each successive convolutional block has double the number of filters of the previous one. The first block has 32 filt.

For each repetition multiple Dense and Transition Layers are stacked, following the pattern [6,12,16].

Fig. 4.10: Flowchart of the proposed DenseNet-like (NN3) architecture.

4.11.3 ResNet-like

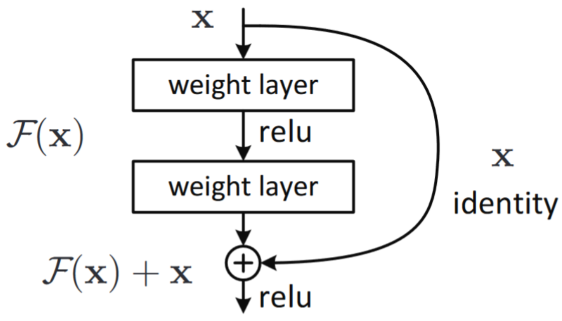


Fig. 4.11: A residual block, showing the skipped connection distinguishing of the ResNet family. Source: *He et al.*, 2016[100].

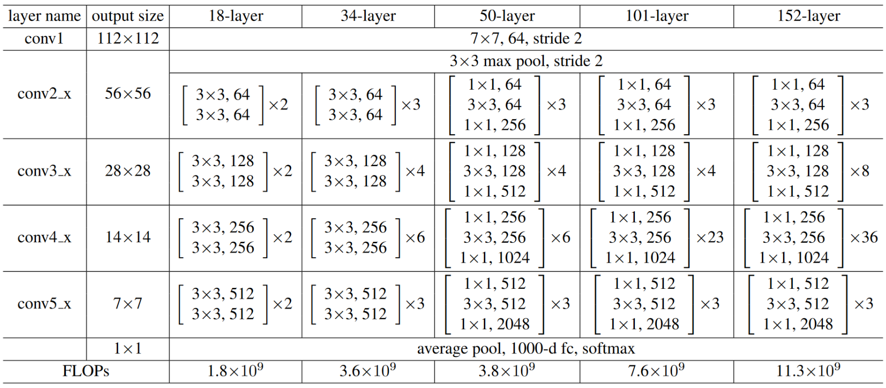
The ResNet (Residual Network) family of neural networks has been primarily employed in computer vision problems, and it has been designed to address the degradation problem encountered when training very deep neural networks. The degradation problem refers to the difficulty of training deep networks as their depth increases. Deeper networks tend to suffer from diminishing accuracy, becoming more challenging to optimize and prone to overfitting. ResNet introduces the concept of residual learning to overcome this issue.

The core building blocks of ResNet are residual blocks, shown in Fig. 4.11. These blocks contain skip connections, shortcut connections or identity mappings, allowing the network to learn residual functions. Instead of directly attempting to learn the desired underlying mapping from the input to the output, residual blocks aim to learn the residual mapping by fitting the difference between the input and output. This helps in training deeper networks more effectively. The skipped connections allow information to bypass one or more layers and be directly fed into deeper layers. Doing so, they mitigate the vanishing gradient problem, which occurs during backpropagation in very deep networks. Skip connections facilitate the flow of gradients through the network, enabling easier optimization of deeper architectures. This enables the possibility of building a very deep network without an excessive increase in the free parameters of the network.

Typically, a global average pooling layer is employed at the end of the network instead of fully connected layers with large numbers of parameters. This pooling layer reduces overfitting and the number of parameters, making the network more computationally efficient. The global average pooling is used in the proposed ResNet-like architecture.

As noted in section 4.9, the proposed architecture upsamples the freestream parameters input and concatenates the resulting tensor to the rakes input. The resulting tensor is used as input to the ResNet network. Additionally, unlike the ResNet networks used for computer vision, the output of the proposed network consists of a fully connected layer and subsequent reshaping of the output to obtain the principal harmonics of the aerodynamic coefficients. The architecture is shown in Fig. 4.9.

The proposed architecture is based on ResNet-34, whose original architecture was first proposed in the original work of He et al. (2016)[100], as shown in Figure 23. As seen in Fig. 4.9, in ResNet-34, the basic residual block consists of two convolutional layers along with a shortcut connection that skips one layer. Thus, the proposed baseline architecture employs two convolutional residual blocks.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. 4.12: ResNet structure defined by the authors in the original paper[100]. Source: *He et al.*, 2016[100].

In contrast, more complex architectures such as ResNet101 employ bottleneck blocks, which consist of three convolutional layers: 1x1, 3x3, and 1x1 convolutions, where the 1x1 convolutions are used to reduce and then restore the dimensions of the input feature maps. The use of bottleneck layers was set as a hyperparameter for ResNets.

4.11.4 DenseNet-like

DenseNet (Densely Connected Convolutional Network) is a deep convolutional neural network architecture based on ResNet, with which it shares the use of skipped connections—known for its densely connected layers, designed to address the challenges of feature propagation and gradient vanishing in very deep networks[101]. Within each dense block, layers share information through densely connected feature maps. The structure of a dense block is shown and further explained in Fig. 4.13. DenseNet architectures typically employ bottleneck layers, consisting of a combination of 1x1 convolutions followed by 3x3 convolutions. This design reduces the number of input feature maps before the 3x3 convolution, which reduces the number of parameters. Between dense blocks, transition layers are used to down-

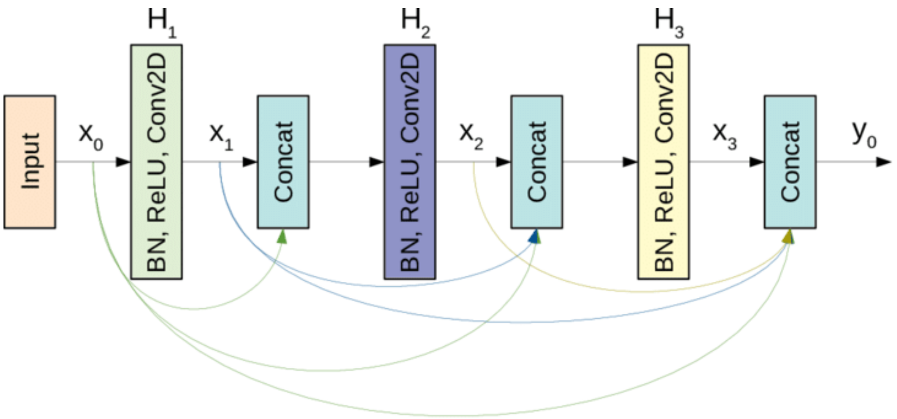


Fig. 4.13: Representation of a DenseBlock. Each layer within the DenseBlock consists of convolutional layers followed by batch normalization and rectified linear unit (ReLU) activation. The skip connections concatenate the feature maps of all previous layers with the current layer's output, allowing each layer to directly access and utilize the feature maps produced by all preceding layers. This architecture improves feature propagation and helps learn hierarchical patterns by feature reuse and improved information flow[101]. Source: *Alshazly et al.*, 2021[102].

sample feature maps by incorporating convolutional layers and pooling operations (e.g., through the use of 1x1 convolutions). These layers help reduce the dimensionality of feature maps while controlling the number of parameters and reducing computational cost. The dense connections mitigate the vanishing gradient problem, enabling the practical training of very deep networks while enabling feature reuse (by skipping connections); DenseNet efficiently utilizes parameters, leading to better parameter efficiency[101].

As noted in section 4.9, the proposed architecture upsamples the freestream parameters input and concatenates the resulting tensor to the rakes input. The resulting tensor is used as input to the DenseNet network. Additionally, unlike the ResNet networks used for computer vision, the output of the proposed network consists of a fully connected layer and subsequent reshaping of the output to obtain the principal harmonics of the aerodynamic coefficients. The architecture is shown in Fig. 4.10

4.12 Models training

The methodology for training the proposed architectures and the subsequent HPO process to improve the performance of the models is presented in the following section. The main choices made in the training process are described, and the reasons for them are discussed. The results of the baseline models before the optimization of the hyperparameters are then shown. The hypermodels and hyperparameters that varied during the HPO process are then defined. A schematic representation of the procedure is shown in Fig. 4.14.

Baseline models were trained on CPU for ease of testing and to compensate for the limited availability of GPU resources. The hyperparameters optimisations were carried out on the high-power nVidia Tesla V100 available at the HPC clusters of the Politecnico di TorinoHPC GPU units, allowing a significant reduction in computation time.

4.12.1 Optimizer

The optimizer used is Adam, with a starting learning rate of 0.005. To make up for the lack of the ability to use an adaptive learning rate and to avoid using the learning rate as a hyperparameter (increasing the size of the search space during the HPO operation), the TensorFlow callback ReduceLROnPlateau was used. It is used to adjust the learning rate during training dynamically. For more information, see section appendix B.

4.12.2 Epochs

The number of training epochs governs the trade-off between the neural network's performance, as measured by the chosen metric (see section 4.5), and the generalization capabilities of the network. That is, generally, as the number of epochs increases, there is an improvement in the selected metric on the training and validation set. On the other hand, an excessive increase in the number of epochs results in poorer performance over the validation set and a loss of network generalization capability, i.e. overfitting. Albeit the risk of overfitting was reduced by taking various precautions (i.e. reducing the number of parameters and regularisation), the number of epochs was limited by early stopping (see appendix B). Then, for each trained model, the number of epochs was chosen by referring to the number of epochs used by networks with similar architecture

Baseline Model	Epochs
FFNN (NN0)	400
Basic CNN (NN1)	750
ResNet-like (NN2)	750
DenseNet-like (NN3)	750

Tab. 4.5: Set training times for baseline models chosen by similar models with similar characteristics.

and number of parameters. Tab. 4.5 shows the number of set training epochs for each of the baseline models. The training was then stopped at the epoch of minimum RMSRE metric one the validation split thanks to Early Stopping callback (for more information see appendix B).

4.12.3 Batch Size

The batch size was determined by testing different values to obtain maximum performance for each model while minimizing the time and resources needed to train. Even if the data sets used for training were small enough to be fully loaded in memory on the computer resources made available by the Politecnico di Torino, it was decided to use a mini-batch size of 32.

4.12.4 Baseline models results

The training results of the baseline models (i.e., the proposed architectures before undergoing the hyperparameter optimization process), divided by dataset and proposed architecture, are shown in Tab. 5.1. The number of training epochs corresponds to the number of epochs for which the minimum value of the loss function on the validation split was obtained via an early stopping callback (see appendix B). Note how, for each of the proposed architectures (and their variations due to different dataset sizes), the optimal number of training epochs was used as the basis for the maximum number of training epochs during the hyperparameter optimization process.

4.13 Hyperparameter optimization

The HPO process was performed using KerasTuner[103], a hyperparameter optimization framework compatible with the TensorFlow and Keras libraries. The algorithm employed to select better-performing models is HyperBand[104], a bandit system optimization algorithm that tests randomly sampled configurations of hyperparameters for a limited number of epochs by early stopping, discards the worst performers and continues training by matching the best performers for a larger number of epochs[104]. The optimization process was evaluated and visualized using TensorBoard, a tool provided by TensorFlow that enables the visualization, debugging, and testing of Keras and TensorFlow models to show the relationship between model performance and preferred hyperparameter combinations[105]. As the

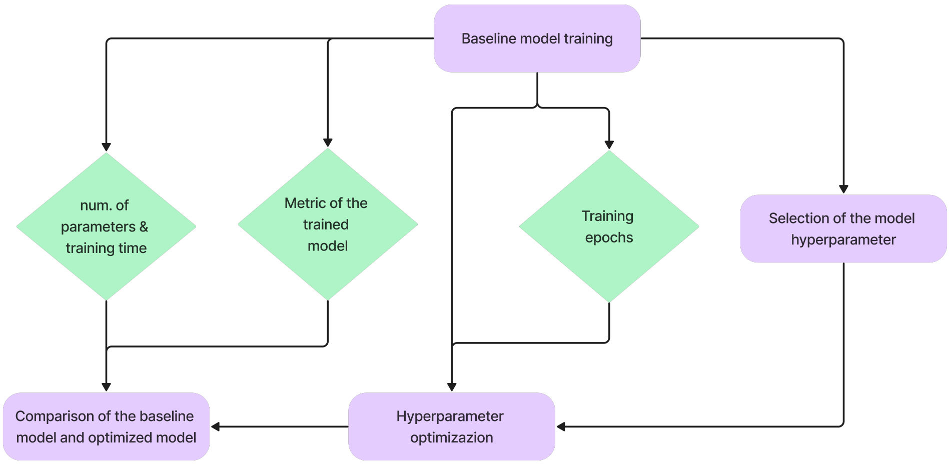

Baseline model training
num. of parameters & training time
Metric of the trained model
Training epochs
Selection of the model hyperparameter
Comparison of the baseline model and optimized model
Hyperparameter optimizazion

Fig. 4.14: Visualization of the hyperparameter optimization workflow. The number of training epochs and metrics of the baseline models were used to define the maximum number of training epochs for each trial (combination of hyperparameters). In contrast, the structure of the baseline model was analyzed to identify a set of hyperparameters (structural and training). The HyperBand algorithm explored the resulting hyperparameter space. For each proposed network, the best-optimized network was compared with the baseline model about training computational cost, training CPU time and model metrics.

algorithm's authors noted, HyperBand can suffer from the bias of selecting poorer-performing models that converge quickly[104]. This was avoided by not considering the learning rate as a hyperparameter (an adaptive learning rate optimizer was chosen; see section 4.12.1).

4.13.1 HyperBand Options

The HyperBand algorithm provides several options to configure and execute the optimization process efficiently. The main options are shown in Tab. 4.6 and are discussed below.

Objective The metric or objective function to optimize during the hyperparameter search. The algorithm's goal is to reduce the value of the loss function as much as possible.

Max Epochs Determines the maximum number of epochs allocated for training each combination of hyperparameters. It utilizes this parameter to limit the training time for each set of hyperparameters. For each hypermodel, this parameter was set at 500 training epochs. Although the baseline models were trained for a maximum number of epochs equal to 750, the maximum number for the HPO process was set equal to 500 for all hypermodels to reduce computational time and cost. This was considered valid because although some baseline models were trained for a maximum of 750 epochs, this was due to the strict early stopping criteria of 100 epochs, and the network improvement in the last 300 epochs was negligible or null.

Factor The reduction factor is applied to the number of configurations and epochs in each successive halving iteration. It controls the ratio by which the configurations are reduced in each round of the algorithm. It sets the total number of epochs of the algorithm's duration according to the formula:

$$(max\ epochs)(\log_{factor}(max\ epochs))^2 \tag{4.12}$$

It has been set to 3.

Iterations The number of times the full HyperBand algorithm is iterated. Higher values lead to better results. It has been set to 2

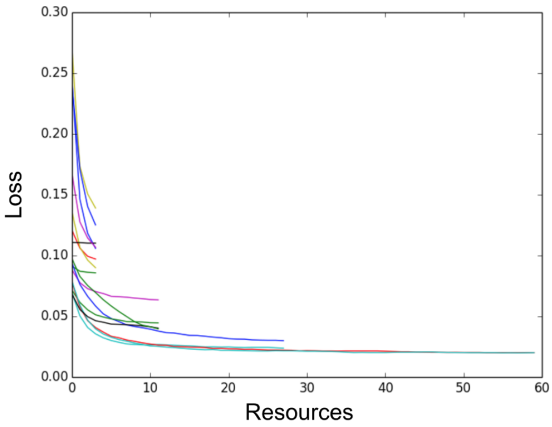


Fig. 4.15: The Hyperband algorithm's iterative process allocates resources to diverse hyperparameter configurations, progressively discarding poorer performers in a bandit-like format and concentrating computational power on promising settings until selecting the best-performing configuration. Source: *Li et al.*, 2018[104].

Max model size Trials with more learnable parameters than specified are rejected and not trained. The value was set to 8000000. Doing so reduced the HPO time by avoiding training networks that would not yield satisfactory results due to the high number of parameters and small dataset size.

Max retries per trial The number of times a specific combination of hyperparameters (trials) can be retried in case the trial crashes. It has been set to 5.

Max consecutive failed trials The termination criterion defines the maximum number of consecutive failed trials that prompt the algorithm to halt. This limit was

Option	Value
Max epochs	500
Factor	3
Iterations	2
Max model size	8000000
Max retries per trial	5
Max consecutive failed trials	150
Executions per trial	3

Tab. 4.6: Values of the main options of the HPO HyperBand algorithm.

fixed at 150 due to occasional invalid combinations of hyperparameters resulting from chosen settings. Setting a high value for this parameter, Max consecutive trials enables the optimization process to persist without manually excluding invalid hyperparameter combinations, albeit at the expense of increased computational time.

Executions per trial The number of times a specific model architecture and set of hyperparameters are evaluated during the hyperparameter tuning process. It allows for a more robust model performance evaluation with a particular hyperparameter. It allows every model to have a fairer evaluation, considering the random differences in trial performance caused by different initialization weights.

4.13.2 List of hyperparameters

Hyperparameters are the internal settings of the model that affect the training process (e.g., learning rate) or the structure of the model (e.g., number of nodes in a fully connected layer). Hyperparameters are external configuration choices that remain constant during training, in contrast to model parameters, which are learned during training (e.g., weights in a neural network). They affect how the model learns, generalizes, and are set before the learning process starts.

For every suggested network, several hyperparameters were found; these are displayed in Tab. 4.7 - 4.10. The hyperparameters and the allowed values during the hyperparameter optimization process are given for each architecture. In the following list, all hyperparameters considered are explained.

Filter Height Kernel height (i.e., the first dimension of 2D kernels) was chosen as the hyperparameter and applied to all hypermodels considered. The default value was considered to be 5 since the first dimension of the tensor rake data encodes the temporal data. Given the low time-step utilized (t_s=0.0002), it was considered possible to use a larger kernel size since the expected variation between two adjacent tensor elements is smaller than expected on the axis that encodes spatial data.

Filter Width Kernel height (i.e., the first dimension of 2D kernels) was chosen as the hyperparameter and applied to all hypermodels considered. The default value was considered 3, smaller than the default kernel height, as the variation of adjacent elements on the second axis of the rake data tensor is expected to be more significant given the distance between rake nodes of 1 cm.

Number of filters in the 1st conv. layer The number of filters in the first convolutional layer governs the number of filters in subsequent layers. For example, in the Basic CNN architecture, each successive convolutional layer has twice as many filters as the previous layer.

Width of the last fully connected layers The number of nodes in the last fully connected layer produces the output of the neural network (then reshaped by a reshape layer in the expected shape 3×2×3).

Pool layer type The type of Pooling used by the network, when provided. Pooling considered are Average Pooling and Max Pooling.

Activation function The activation function used by fully connected layers and, when provided, by convolutional layers. Three activation functions were considered: ReLU, GELU, and SWISH.

Number of convolutional layers The number of convolutional layers used in the network. This hyperparameter was applied only to the Basic-CNN network.

Number of deconvolutional layers The number of deconvolutional layers used for upscaling of scalar parameters. See section 4.9 for more information.

Number of dense layers The number of fully connected layers downstream of the tensor data flattening used before the output layer. To reduce the number of hyperparameters, each fully connected layer has the same number of nodes, governed by the hyperparameter ”Width of the last fully connected layer.”

L2 regularization value As previously mentioned, L2 regularization reduces the model weights' absolute value. This is achieved by adding an extra term to the loss function.

$$\lambda_r \sum_{j=1}^{p} \beta_j^2 \tag{4.13}$$

The hyperparameter is thus the value of the regularization parameter λ_r.

Dropout rate The dropout rate is the probability of dropping out neurons in a neural network layer during training. The hyperparameter determines the fraction or percentage of neurons that will be randomly ignored or ”dropped out” during each training iteration.

Stride size The stride size is the number of tensor elements (or pixels, by analogy with a 2D picture) by which the kernel moves across the input image or feature map.

Number of Residual Blocks repetitions The ”dense block repetitions” parameter signifies how many times residual blocks are stacked or repeated in the overall architecture of the ResNet. A higher number of repetitions generally allows for deeper and more complex feature extraction within each block, potentially capturing more intricate patterns in the data. However, it also increases the model's parameters and computational requirements. As the information encoded in the datasets is relatively simple, fewer repetitions are expected to fare better. The number of repetitions has been chosen to comply with the number used in the original ResNets variants (ResNet18, ResNet34, ResNet50, ResNet101, ResNet152)[100].

Bottleneck layer in Convolutional Block Only employed in the ResNet-like architecture controls whether a bottleneck layer is employed in the network. The bottleneck layer typically consists of a 1x1 convolutional layer responsible for reducing the dimensionality of the input feature maps and a successive convolutional layer (whose kernel size is governed by the hyperparameters ”Filter Width” and ”Filter Height”). Then, another 1x1 convolutional layer expands the feature maps back to their original dimensions. This expansion helps maintain the network’s representational power[100]. The bottleneck layer is usually employed in the more complex ResNet variants, such as ResNet101 and ResNet152[100]. For additional information, see section 4.11.3.

DenseBlock employs activation function at end Controls whether or not an activation function is used at the output of a DenseBlock. The activation function used is governed by the hyperparameter ”Activation Function”.

Number of DenseBlock repetitions The ”dense block repetitions” parameter signifies how often these dense blocks are stacked or repeated in the overall architecture of the Dense Net. A higher number of repetitions generally allows for deeper and more complex feature extraction within each block, potentially capturing more intricate patterns in the data. However, it also increases the model’s parameters and computational requirements. As the information encoded in the datasets is relatively simple, fewer repetitions are expected to fare better.

Tensor flattening To process and prepare the convolutional layer output for the final fully connected output layer, the tensor data has to be flattened into a 1-D array. Flattening achieves this, that is, by reshaping a multi-dimensional tensor (such as a 2D image or a higher-dimensional feature map) into a 1D vector while maintaining the order of elements or by Global Average Pooling that computes the average value for each feature map channel across the spatial dimensions of the feature map. The usage of one of these layers has been set up as a hyperparameter.

Hyperparameters (NN 1)	Value
Number of filters in the 1^{st} conv. layer	[8, 16, 32, 64]
Width of the last fully connected layers	[512, 1024, 2048, 4096]
Filter Height	[3, 5, 7]
Filter Width	[3, 5, 7]
Pool layer type	[Average, Max]
Activation function	[GELU, ReLU, SWISH]
Number of convolutional layers	[3,5,7,9]
Number of deconvolutional layers	[1,2,3]
Number of dense layers	[1,2,3]
L2 regularization value	[0, 0.00025, 0.00050, 0.00075, 0.00100]
Dropout rate	[0.0, 0.3, 0.5, 0.7]

Tab. 4.7: Hyperparameters chosen for the Basic CNN network optimization process (NN1).

Hyperparameters (NN 2)	Value
Number of filters in the 1^{st} conv. layer	[16, 32, 64, 128]
Width of the last fully connected layers	[512, 1024, 2048, 4096]
Filter Height	[3, 5, 7]
Filter Width	[3, 5, 7]
Activation function	[GELU, ReLU, SWISH]
Number of deconvolutional layers	[1,2,3,4]
Pool layer type	[Average, Max]
Tensor Flattening	[Flattening, Global Avg. 2D Pool.]
Stride size	[2, 3]
Number of dense layers	[1,2,3,4]
L2 regularization value	[0, 0.00025, 0.00050, 0.00075, 0.00100]
Dropout rate	[0.0, 0.3, 0.5, 0.7]
Number of Residual Blocks repetitions	$\begin{bmatrix} [2,2,2,2], & [3,4,6,3], \\ [3,4,23,3], & [3,8,36,3] \end{bmatrix}$

Tab. 4.8: Hyperparameters chosen for the ResNet-like network optimization process (NN2).

Hyperparameters (NN 3)	Value
Number of filters in the 1st conv. layer	[16, 32, 64, 128]
Width of the last fully connected layers	[512, 1024, 2048, 4096]
Filter Height	[3, 5, 7]
Filter Width	[3, 5, 7]
Activation function	[GELU, ReLU, SWISH]
Number of deconvolutional layers	[1,2,3,4]
Number of dense layers	[1,2,3,4]
L2 regularization value	[0, 0.00025, 0.00050, 0.00075, 0.00100]
Dropout rate	[0.0, 0.3, 0.5, 0.7]
Stride size	[2, 3]
DenseBlock employs activation function at end	[TRUE, FALSE]
Number of DenseBlock repetitions	$\begin{bmatrix} [6,12], & [6,12,24,16], \\ [6,12,24], & [6,12,32,32], \\ [6,12,32], & [6,12,48,32], \\ [6,12,48], & \\ [6,12,64] & \end{bmatrix}$

Tab. 4.9: Hyperparameters chosen for the DenseNet-like network optimization process (NN3).

Hyperparameters (NN 0)	Value
Width of the fully connected layers	[512, 1024, 2048, 4096]
Activation function	[GELU, ReLU, SWISH]
Number of fully connected layers	[1,2,3,4,5,6,7,8,9,10]
L2 regularization value	[0, 0.0025, 0.0050, 0.0075, 0.00100]
Dropout rate	[0.0, 0.3, 0.5, 0.7]

Tab. 4.10: Hyperparameters chosen for FFNN network optimization process (NN0). Due to the lower complexity of the network, the number of chosen hyperparameters is lower.

Chapter

Results 5

In this chapter, the baseline model training results are shown and discussed. In addition, the results of the optimisation of the hyperparameters are shown, and the metrics of the optimised models are compared with the metrics of the baseline models to establish the effectiveness of the HPO procedure.

5.1 Results of the baseline models

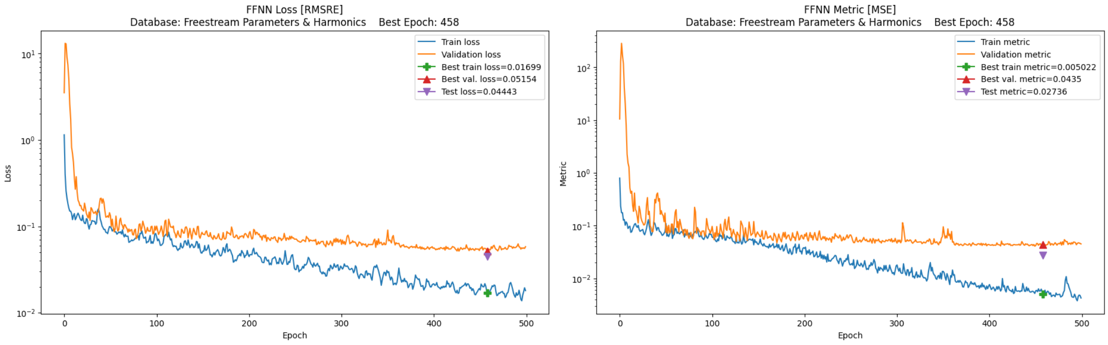
Training results for all networks considered and matched datasets are shown in Tab. 5.1 and the loss and metric value trends over training can be seen in Fig. 5.1 - 5.4.

All networks achieved similar results, both in terms of loss and metric value. Probably due to the limited size of the dataset, all networks achieved saturation; having extracted all relevant information from the small datasets has extracted all the relevant information the dataset, it is often described as having reached saturation or having achieved the maximum capacity for the given dataset. This suggests that the complexity of the problem requires a larger amount of data for training. Indeed, some preliminary testing performed joining the train and test dataset (effectively increasing the size of the training dataset by 10%) showed improved results. The results were not included because the absence of test data does not allow for evaluating the generalizing network well to unseen data.

The network that obtained better results is the Basic CNN, which has a computational cost of training and reduced training time compared to other convolutional networks. In addition, convolutional networks using the DS1 Wake rake dataset obtained better results. This could be due to the fact that the dataset includes only the most representative results concerning the wake. This is in contrast to what was initially expected: that the DS1 Box rake would allow the networks to have a better predictive ability due to the larger amount of near-field data collected.

Fig. 5.1 and Tab. 5.1 show that the FFNN network achieved comparable results to convolutional networks, but CNNs showed better generalization capabilities on the validation and test datasets. The better generalization capabilities of the CNN networks are probably due to the additional near-field information collected on the rakes and the better resistance of the CNNs to overfitting. The similar performances suggest that the chosen baseline architectures either do not take full advantage of the additional near field information encoded in the DS1, DS2 and DS3 datasets or that the fluid dynamic field data are too sparse to provide the network additional information to gain better prediction capabilities. Indeed, similar works that have used convolutional networks to predict airfoil performance have used the entire flow field to gain better network performances[9]. It is also worth noting that similar works that employed neural networks to predict the aerodynamic performance of airfoils in the transonic field achieved comparable network predicting capabilities[106].

It can be seen that some networks converged to the maximum number of training epochs. This was not considered problematic and may be caused by the strict patience value for the early stopping callback of 100 epochs. Indeed, for networks trained for the maximum number of epochs, the gain on validation loss in the last 400 epochs is negligible or null.



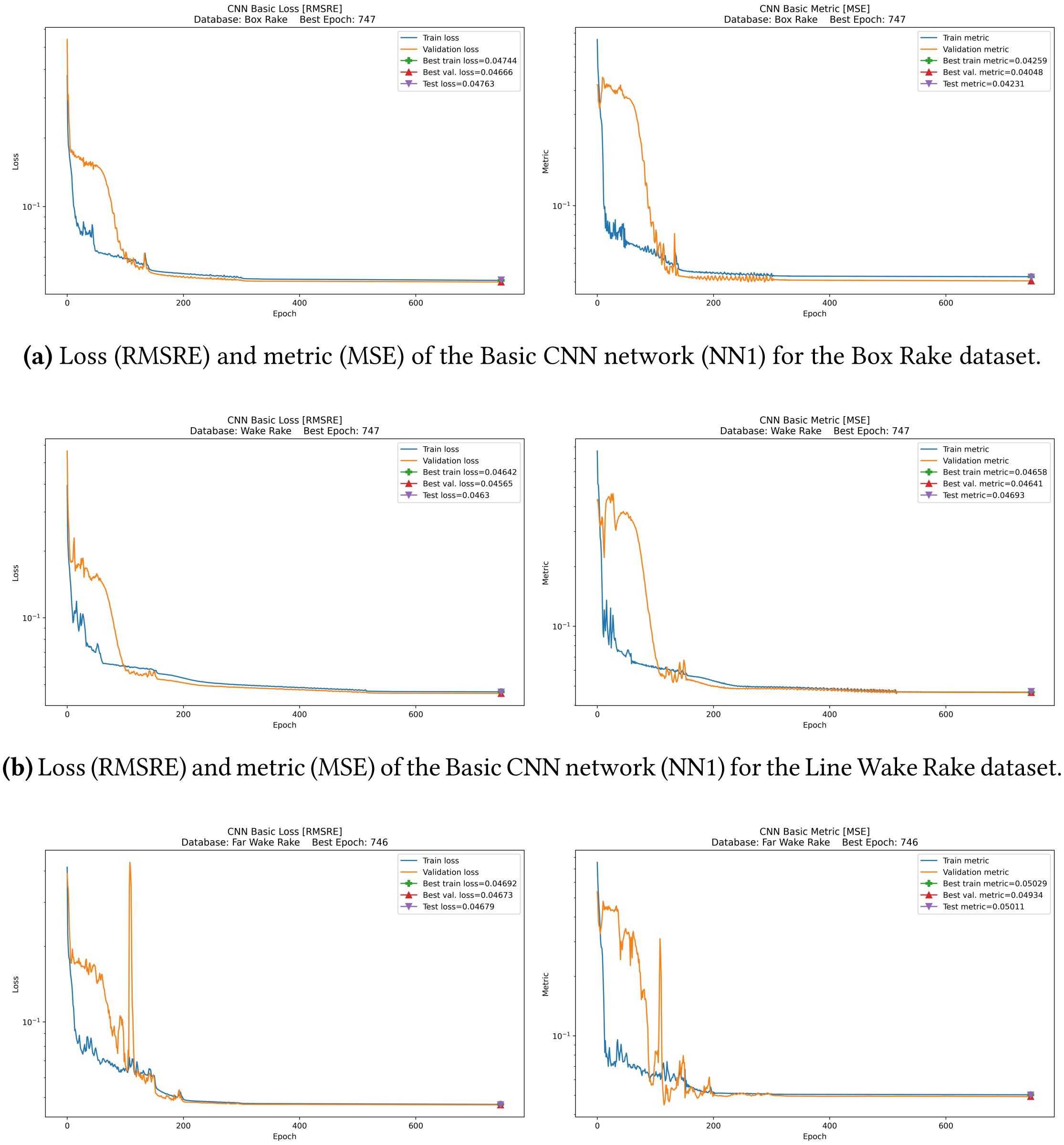
(a) Loss of the FFNN network (NN0).

(b) Metric of the FFNN network (NN1).

Fig. 5.1: Loss value trends during the training of the FFNN (NN0) network.

	Baseline Model	FFNN (NN0)	Basic CNN (NN1)	ResNet-like (NN2)	DenseNet-like (NN3)
	Parameters	206,098	426,123	2,963,403	3,886,283
Dataset 0	**Epochs**	458	-	-	-
	Train Loss	0.01699	-	-	-
	Val. Loss	0.05154	-	-	-
	Test Loss	0.04443	-	-	-
	Train Metric	0.05022	-	-	-
	Val. Metric	0.04350	-	-	-
	Test Metric	0.02736	-	-	-
Dataset 1	**Epochs**	-	747	744	583
	Train Loss	-	0.04744	0.05858	0.02634
	Val. Loss	-	0.04666	0.0562	0.03907
	Test Loss	-	0.04763	0.0518	0.0301
	Train Metric	-	0.04259	0.05073	0.0740
	Val. Metric	-	0.04048	0.04955	0.07399
	Test Metric	-	0.04231	0.04977	0.07318
Dataset 2	**Epochs**	-	747	442	213
	Train Loss	-	0.04642	0.05613	0.04830
	Val. Loss	-	0.04565	0.05340	0.05000
	Test Loss	-	0.04630	0.05569	0.04990
	Train Metric	-	0.04658	0.0500	0.04414
	Val. Metric	-	0.04641	0.05078	0.04218
	Test Metric	-	0.04693	0.04993	0.04378
Dataset 3	**Epochs**	-	746	742	164
	Train Loss	-	0.04692	0.05507	0.04988
	Val. Loss	-	0.04673	0.05230	0.05448
	Test Loss	-	0.04679	0.05519	0.05187
	Train Metric	-	0.05029	0.05001	0.04417
	Val. Metric	-	0.04934	0.04982	0.04772
	Test Metric	-	0.05011	0.05153	0.04672

Tab. 5.1: Performance of the baseline models (before HPO) for each of the 3 datasets considered. The training epoch achieving the best possible validation loss is also shown.

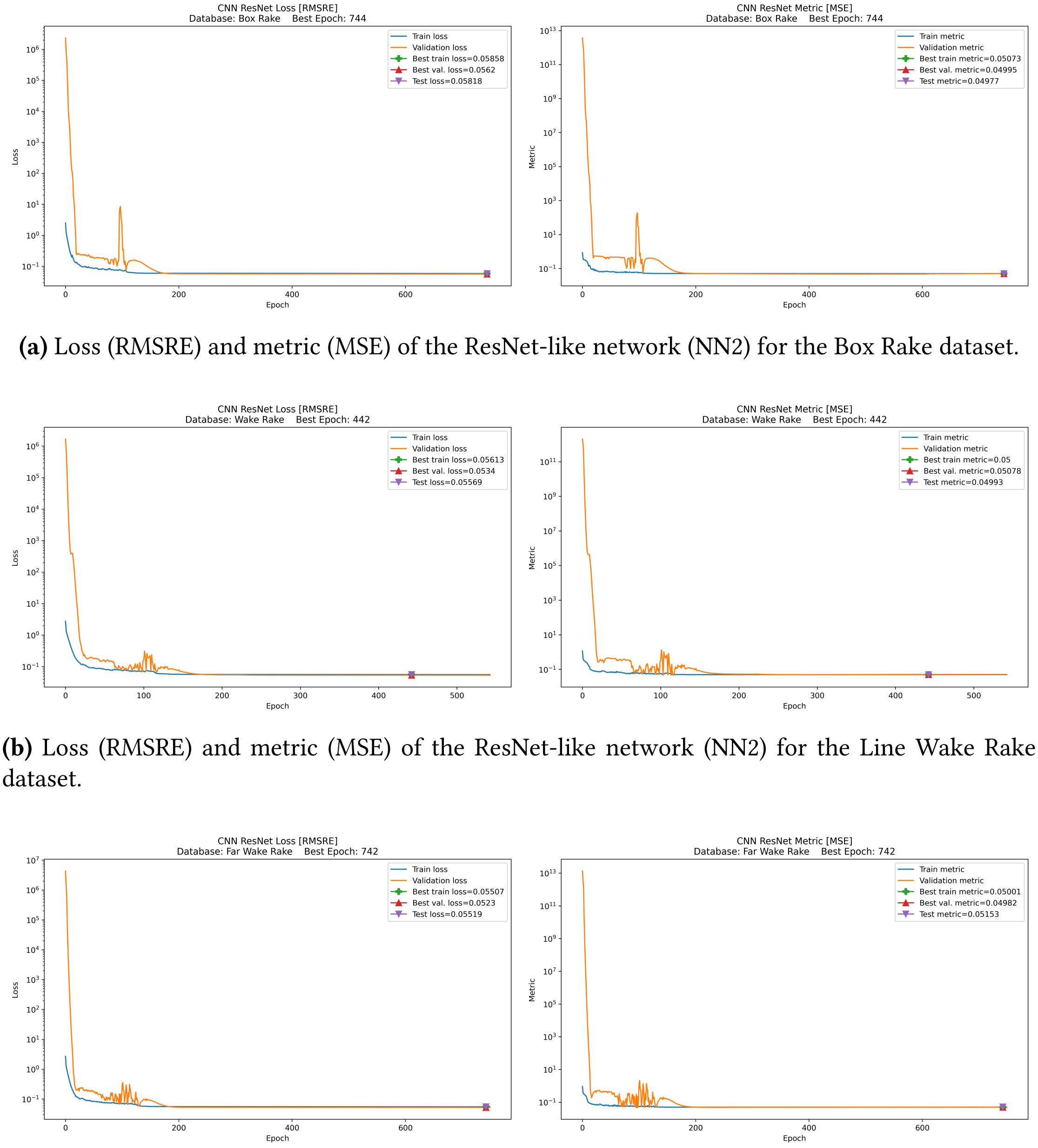


(a) Loss (RMSRE) and metric (MSE) of the Basic CNN network (NN1) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the Basic CNN network (NN1) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the Basic CNN network (NN1) for the Far Wake Rake dataset.

Fig. 5.2: Loss and metric value trends during the training of the Basic CNN (NN1) network for all the Datasets.

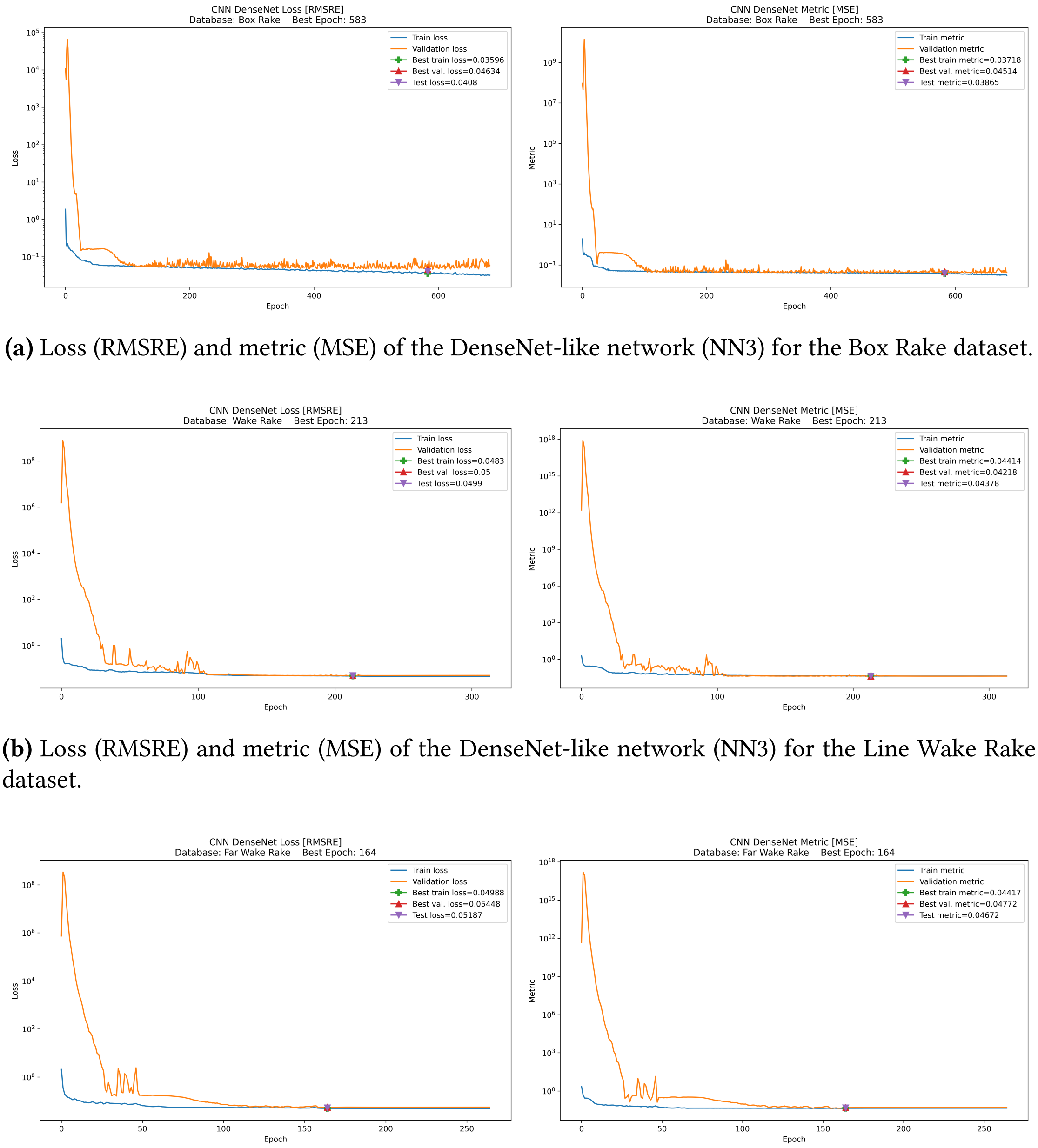


(a) Loss (RMSRE) and metric (MSE) of the ResNet-like network (NN2) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the ResNet-like network (NN2) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the ResNet-like network (NN2) for the Far Wake Rake dataset.

Fig. 5.3: Loss and metric value trends during the training of the ResNet-like network (NN2) network for all the Datasets.



(a) Loss (RMSRE) and metric (MSE) of the DenseNet-like network (NN3) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the DenseNet-like network (NN3) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the DenseNet-like network (NN3) for the Far Wake Rake dataset.

Fig. 5.4: Loss and metric value trends during the training of the DenseNet-like network (NN3) network for all the Datasets.

5.2 Optimized hyperparameters and optimized model results

The hyperparameters of the optimized networks are shown in Tab. 5.3 - 5.6. Training results for all the optimized networks are shown in Tab. 5.2 and the loss and metric value trends over training can be seen in Fig. 5.5 - 5.8.

The optimized networks achieved similar performance to the baseline models. This suggests that the networks, regardless of the values of the hyperparameters, have achieved the maximum predictive capabilities allowed by the dataset and that better results can only be obtained by increasing the size of the training datasets.

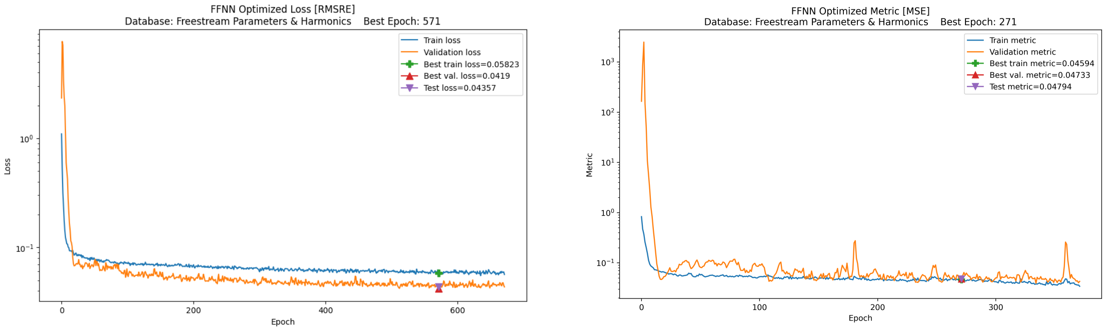
Since all networks achieved comparable results, the bandit-like system of the HyperBand HPO optimizer selected models with fewer parameters and shorter training times.

Hyperparameter values for the Basic CNN optimized networks are given in Tab. 5.3. Every network variant employs fewer filters, possibly to reduce the number of trainable parameters of the model. Moreover, all the network variants selected the SWISH or GELU activation function. The SWISH activation function has been shown to reduce training time in deeper networks[63]; however, the optimized networks have fewer convolutional layers (possibly to reduce training time): this could show, as hypothesized, that the GELU and SWISH activation functions can be used as drop-in substitutes of ReLU activation. Furthermore, for all variants of Basic CNNs, the HyperBand optimizer selected Global Average Pooling for flattening the feature maps, once again prioritizing lowering the number of model parameters. It should also be noted how all Basic CNN networks exhibit unitary stride and low dropout rate and L2 regularization values: low values of the regularization hyperparameters could be traced to the fact that convolutional networks are known to be resistant to overfitting.

Hyperparameter values for the ResNet-like optimized networks are given in Tab. 5.4. Every network variant employs fewer filters, possibly to reduce the number of trainable parameters of the model. Moreover, all the network variants selected the SWISH activation function. This is likely due to the deepness of the network, as the SWISH activation function has been shown to reduce training time in deeper networks[63]. Finally, all variants selected the Global Average 2D pooling and a low number of ResNet Block repetitions. This is probably because HyperBand optimized for smaller, lower training time models and that models using Flattening

Layer and a higher number of ResNet block repetitions were discarded due to the limit of trainable parameters set to 8000000.

Hyperparameter values for the DenseNet-like optimized networks are given in Tab. 5.5. Every network variant employs fewer filters, possibly to reduce the number of trainable parameters of the model. Moreover, all the network variants selected the GELU or ReLU activation function. Finally, all variants selected the Global Average 2D pooling and a low number of DenseNet Block repetitions. This is probably because HyperBand optimized for smaller, lower training time models and that models using Flattening Layer for the flattening of the feature maps and a higher number of ResNet block repetitions were discarded due to the limit of trainable parameters set to 8000000.



(a) Loss of the optimized FFNN network (NN0).

(b) Metric of the optimized FFNN network (NN0).

Fig. 5.5: Loss value trends during the training of the optimized FFNN (NN0) network.

Optimized Model		FFNN (NN0)	Basic CNN (NN1)	ResNet-like (NN2)	DenseNet-like (NN3)
Dataset 0	**Parameters**	53,906	-	-	-
	Epochs	571	-	-	-
	Train Loss	0.05823	-	-	-
	Val. Loss	0.0419	-	-	-
	Test Loss	0.04357	-	-	-
	Train Metric	0.06153	-	-	-
	Val. Metric	0.04479	-	-	-
	Test Metric	0.04397	-	-	-
Dataset 1	**Parameters**	-	109,107	24,027	157,643
	Epochs	-	458	215	150
	Train Loss	-	0.04940	0.05051	0.1368
	Val. Loss	-	0.04734	0.05045	0.1428
	Test Loss	-	0.05264	0.05129	0.1334
	Train Metric	-	0.05283	0.04383	0.3239
	Val. Metric	-	0.04922	0.04898	0.3746
	Test Metric	-	0.04462	0.04738	0.3214
Dataset 2	**Parameters**	-	796,603	682,635	2,483,417
	Epochs	-	211	426	376
	Train Loss	-	0.04295	0.05569	0.05707
	Val. Loss	-	0.04492	0.05132	0.05803
	Test Loss	-	0.04297	0.05132	0.0805
	Train Metric	-	0.04133	0.04853	0.04143
	Val. Metric	-	0.04574	0.04325	0.04385
	Test Metric	-	0.04349	0.03706	0.03809
Dataset 3	**Parameters**	-	2,415,507	407,963	411,291
	Epochs	-	174	426	428
	Train Loss	-	0.04092	0.04004	0.04635
	Val. Loss	-	0.03968	0.046675	0.04647
	Test Loss	-	0.03925	0.05147	0.0625
	Train Metric	-	0.04117	0.03592	0.04184
	Val. Metric	-	0.04141	0.04277	0.04587
	Test Metric	-	0.03791	0.04686	0.07995

Tab. 5.2: Table summarizing the performance of models optimized by HPO. The loss values, metrics values, and number of training epochs for each combination of the proposed model and dataset are shown.

Hyperparameter (NN1)	Values		
	DS1	DS2	DS3
Number of filters in the 1st conv. layer	16	8	64
Width of the last fully connected layers	1024	512	512
Filter Height	3	3	3
Filter Width	5	3	3
Pool layer type	Avg.	Avg.	Avg.
Activation function	SWISH	GELU	SWISH
Tensor Flattening	Gl. Avg. 2D Pool.	Gl. Avg. 2D Pool.	Gl. Avg. 2D Pool.
Number of convolutional layers	3	6	4
Number of deconvolutional layers	3	3	3
Stride size	1	1	1
Number of dense layers	1	1	1
L2 regularization value	0	0	0
Dropout rate	0.3	0	0

Tab. 5.3: Hyperparameter values of the optimized Basic CNN network (NN1).

Hyperparameter (NN2)	Values		
	DS1	DS2	DS3
Number of filters in the 1st conv. layer	8	8	16
Width of the last fully connected layers	512	512	218
Filter Height	5	5	5
Filter Width	5	3	5
Activation function	SWISH	SWISH	SWISH
Number of deconvolutional layers	3	3	3
Stride size	3	3	3
Pool layer type	Max	Average	Max
Tensor Flattening	Gl. Avg. 2D Pool.	Gl. Avg. 2D Pool.	Gl. Avg. 2D Pool.
Number of dense layers	2	2	1
L2 regularization value	0	0.0001	0.0001
Dropout rate	0	0	0
Number of Residual Blocks repetitions	[2, 2, 2, 2]	[2, 4, 4, 2]	[2, 2, 2, 2]

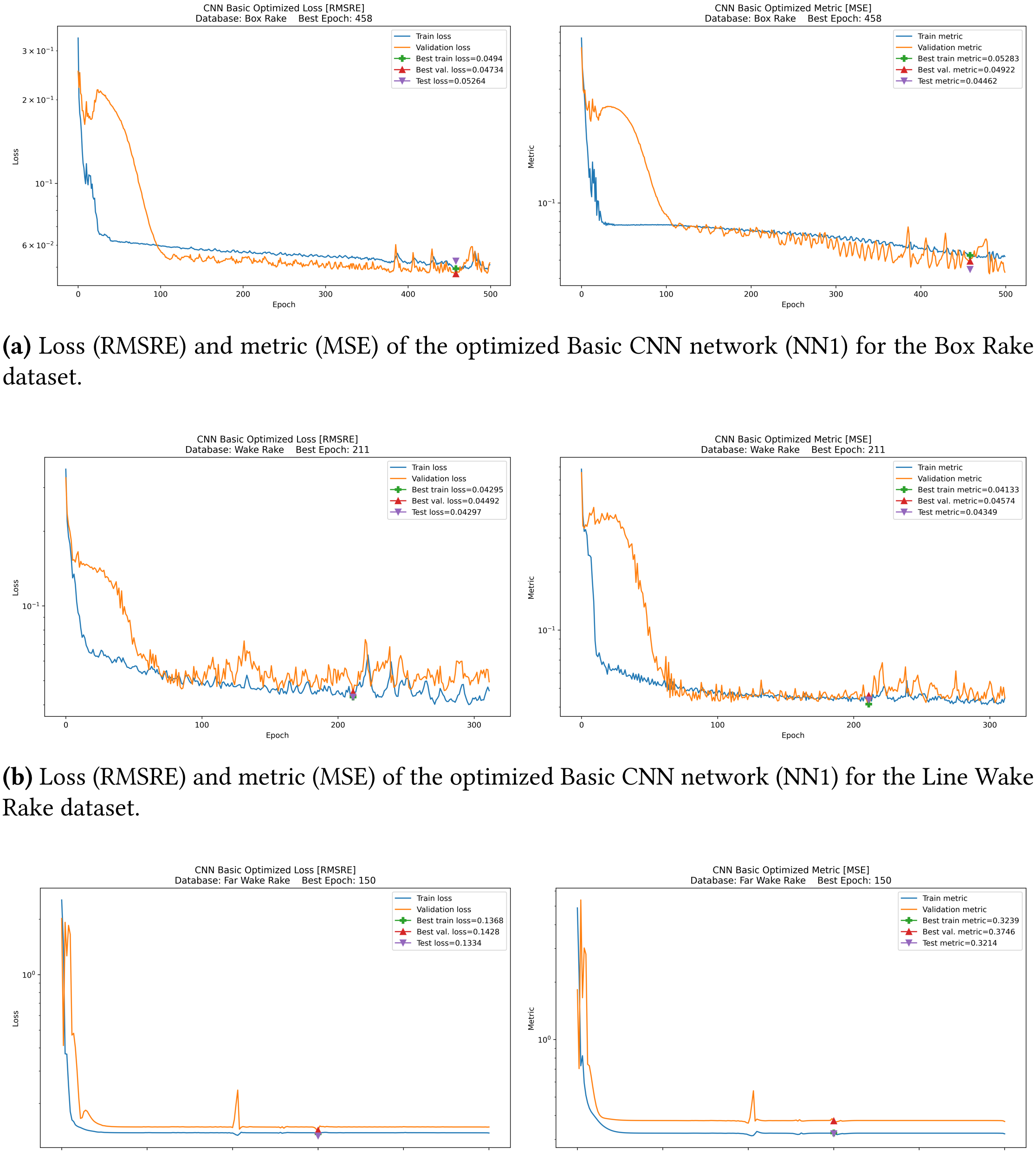
Tab. 5.4: Hyperparameter values of the optimized ResNet-like network (NN2).

Hyperparameter (NN3)	Values		
	DS1	DS2	DS3
Number of filters in the 1st conv. layer	16	8	8
Width of the last fully connected layers	1024	1024	512
Filter Height	3	3	3
Filter Width	5	5	3
Activation function	GELU	GELU	ReLU
Number of deconvolutional layers	3	3	3
Number of dense layers	1	1	2
L2 regularization value	0	0.0001	0.00005
Stride size	3	2	2
Number of DenseBlock repetitions	[6, 12, 32]	[6, 12, 16]	[6,12]

Tab. 5.5: Hyperparameter values of the optimized DenseNet-like network (NN3).

Hyperparameters	Values
	DS0
Width of the fully connected layers	128
Activation function	ReLU
Number of fully connected layers	4
L2 regularization value	0
Dropout rate	0.5

Tab. 5.6: Hyperparameter values of the optimized FFNN network (NN0).

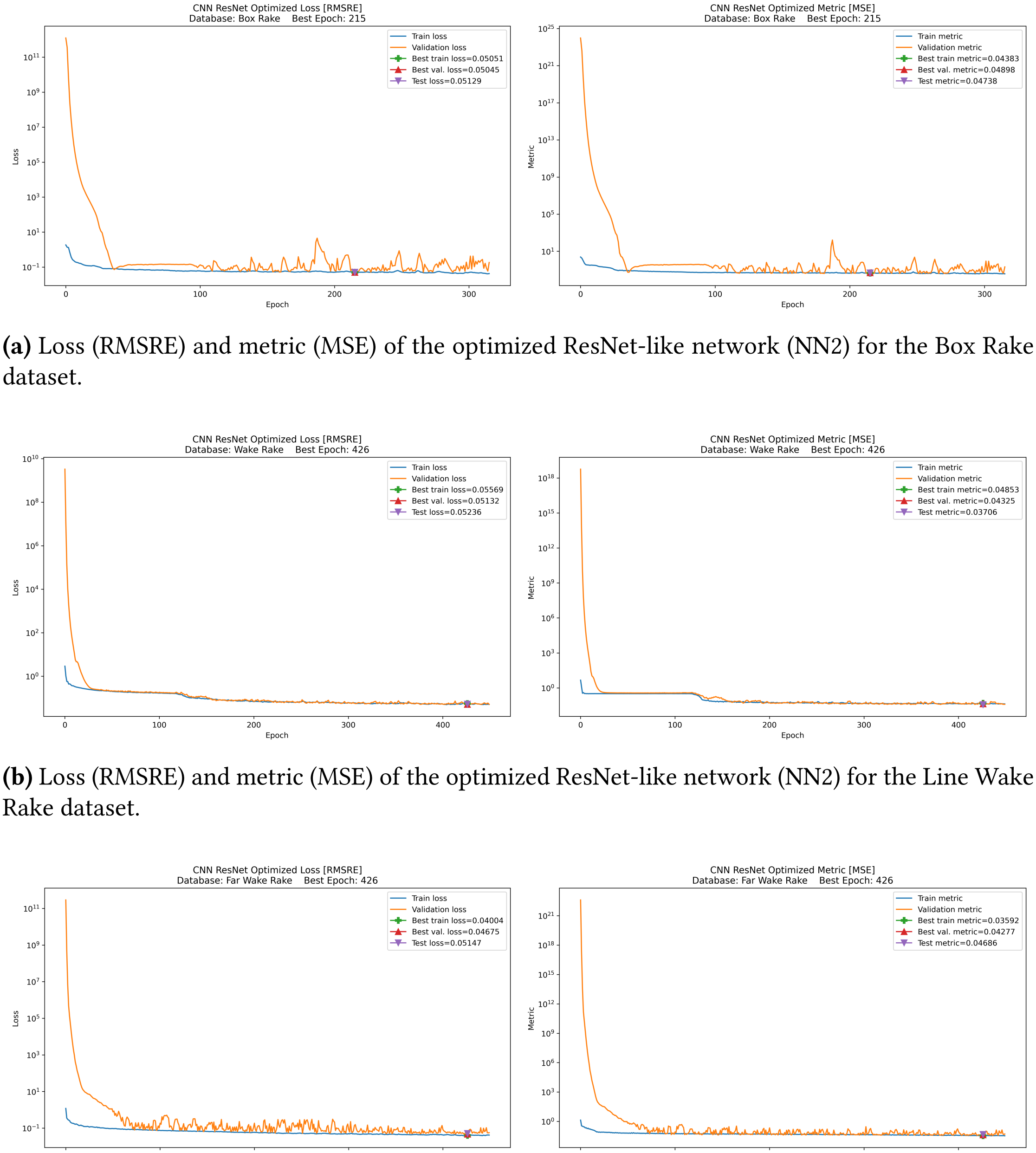


(a) Loss (RMSRE) and metric (MSE) of the optimized Basic CNN network (NN1) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the optimized Basic CNN network (NN1) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the optimized Basic CNN network (NN1) for the Far Wake Rake dataset.

Fig. 5.6: Loss and metric value trends during the training of the optimized Basic CNN (NN1) network for all the Datasets.

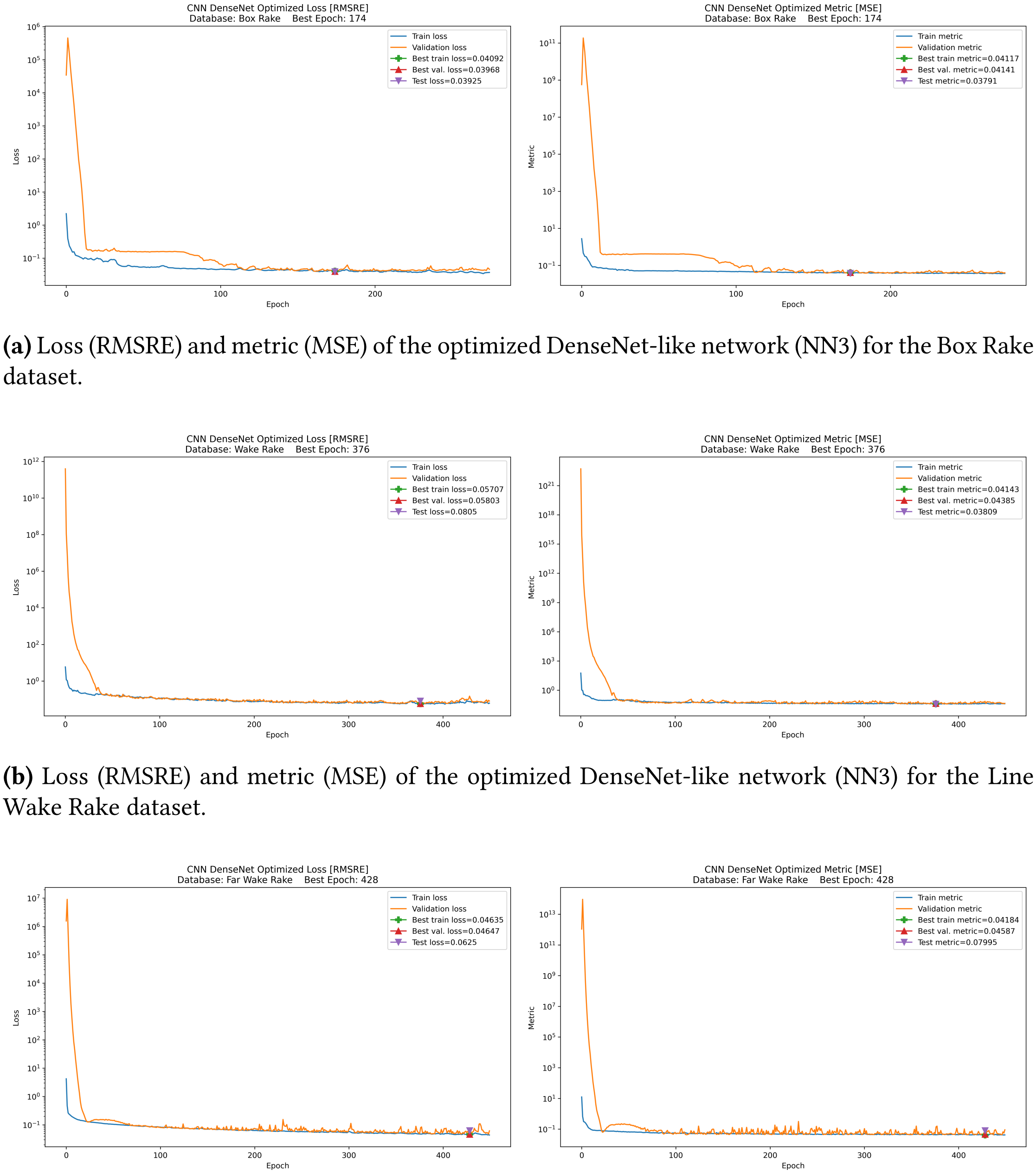


(a) Loss (RMSRE) and metric (MSE) of the optimized ResNet-like network (NN2) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the optimized ResNet-like network (NN2) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the optimized ResNet-like network (NN2) for the Far Wake Rake dataset.

Fig. 5.7: Loss and metric value trends during the training of the optimized ResNet-like network (NN2) network for all the Datasets.



(a) Loss (RMSRE) and metric (MSE) of the optimized DenseNet-like network (NN3) for the Box Rake dataset.

(b) Loss (RMSRE) and metric (MSE) of the optimized DenseNet-like network (NN3) for the Line Wake Rake dataset.

(c) Loss (RMSRE) and metric (MSE) of the optimized DenseNet-like network (NN3) for the Far Wake Rake dataset.

Fig. 5.8: Loss and metric value trends during the training of the optimized DenseNet-like network (NN3) network for all the Datasets.

Chapter

Conclusions 6

The possibility of applying convolutional neural networks to predict aerodynamic performance and near-field flow data of an airfoil immersed in a freestream flow in the highly compressible/transonic regime has been investigated.

Firstly, different datasets were obtained by simulating the airfoil in a 2D domain in the commercial code ANSYS Fluent by varying the scalar values of the freestream parameters governing the phenomenon under investigation, i.e. AoA, Re, M.

We considered airfoil performance prediction networks with input from the near-field flow data collected near the airfoil and additional scalar inputs. Several architectures based on well-known image classification neural networks have been proposed. The models attempted to estimate the magnitudes and frequencies of the 3 dominant harmonics of the time series of the nondimensional airfoil coefficients (i.e. Cd, Cl, Cm) using near-field data and additional input scalar values. Each proposed model was adjusted for 3 different datasets, which differed in the number and collection position of the near-field data. Each variation of the proposed models was then trained and evaluated.

All baseline models achieved similar performance, with relative errors (RMSRE) on the order of 4-5%. This could be due to the small size of the dataset, which has only about 200 training samples. Moreover, CNN networks did not perform significantly better training loss performance than the FFNN, but they showed better generalization capabilities on the validation and test datasets. The better generalization capabilities of the CNN networks are probably due to the additional near-field information collected on the rakes and the better resistance of the CNNs to overfitting. Therefore, better performance of the networks can be achieved by increasing the size of the dataset or by reducing the variation of scalar parameter values (i.e., AoA, Re, M) during the parameter sweep.

Next, each of the proposed models was subjected to HPO using the HyperBand algorithm. During the HPO process, several hyperparameters, such as the number of filters, network depth, activation function, etc., were varied. Each model instance trained during the HPO process was evaluated for multiple iterations to avoid excessive influence of the randomness of the network initialization, which can affect the network training process, especially for small datasets. This ensured a fair evaluation of each model and the effect of each hyperparameter considered.

Hyperparameter optimization led to unsatisfactory results, with the networks optimized with hyperparameters chosen by the HyperBand hyperparameter optimizer achieving comparable performance to the baseline models, in which hyperparameters were selected by limited trial and error and an educated guess. HyperBand selected hyperparameters resulting in smaller, fewer-parameters models with less training time and computational cost and with similar, and in some cases worse, model performance than the baseline models. This is probably because, due to the size of the training datasets the networks generally achieved the best possible performance, given the size of the datasets.

6.1 Future Works

As proof of concept, this paper presents a possible framework for constructing deep-learning models to predict the near-flow field and aerodynamic performance of airfoils.

The present work could be expanded upon by including a higher-accuracy dataset, either acquired from experimental data or high-accuracy simulations. Furthermore, the effect of a more spatially sparse dataset to simulate the actual data collection possibilities in an experimental wind tunnel setup could be explored.

Furthermore, the effect of the airfoil shape could be investigated, as initially planned in the present work, but it is not possible to study due to time and cost constraints. The proposed alternatives are the PARSEC parametrization system[108] (shown in Fig. 6.1) or a binary representation of the airfoil shape as an image, to be fed as input to a neural network convolutional layers. Shape parametrization has been long used during the design and aerodynamic characterization of airfoils, and different similar works in the literature employ parametrization to represent airfoil shape[109]; however, as the PARSEC system and similar ones employ 11 parameters, it can be argued that it could add excessive complexity to the neural network. Indeed, it was initially considered to employ an image representation of

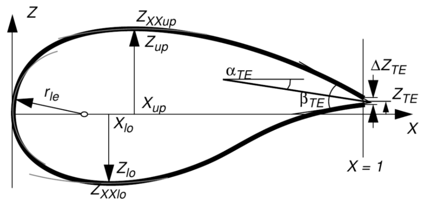


Fig. 6.1: Visualisation of the parsec parametrisation system and its parameters. Source: *Arias-Montano et al.*, 2011[107].

the airfoil shape; several similar works in the literature employ an image representation[110,111]; moreover, such a representation adds the possibility of novel approaches, such as image augmentation, dataset augmentation, or the use of Signed Distance Function to represent the geometry.

In addition, the use of transfer learning, i.e., pre-trained and fine-tuned models for the task at hand (both regression and classification tasks)[28], was considered to compensate for the small size of the dataset. In this specific case, the use of pre-trained networks for image classification was considered: the hypothesis was that the feature detection capability of image classification networks could be exploited by fine-tuned layers at the bottom of the network to extract relationships in the data between the features detected by the pre-trained network to obtain the output data, allowing an increase in performance when applied to a task with a small training dataset[22,26,28] or, more generally, when confronted with a similar task[29]. At present, no work is known that has utilized transfer learning for the prediction of airfoil performance. The use of transfer learning was not included in this work due to unsatisfactory results.

Appendix

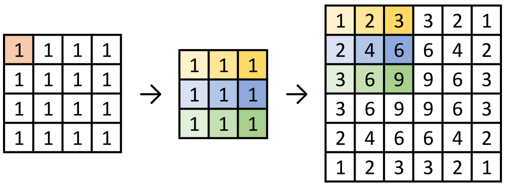
A TensorFlow Layers

In TensorFlow, deep-learning models are composed of layers: reusable functions that accept a tensor input and provide a tensor output, with a defined structure and trainable parameters[58]. Some of the layers adopted in the proposed architectures are presented and discussed, while the design choices behind their adoption are motivated.

Convolutional

As previously mentioned, the data collected on the near-airfoil envelope were encoded in an image-like format where, in analogy to a colour image, different physical quantities are stored on different channels (see section 4.3). Consequently, it was decided to use CNN architectures for analysing the data and extracting spatio-temporal relations within the input data; convolutional layers are therefore employed in the proposed architectures. In particular, the Conv2D layer is adopted, which performs the convolution operation on 4-D tensors and whose operation was illustrated in section 2.5.

Transposed Convolution



$$\begin{matrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{matrix} \rightarrow \begin{matrix}1&1&1\\1&1&1\\1&1&1\end{matrix} \rightarrow \begin{matrix}1&2&3&3&2&1\\2&4&6&6&4&2\\3&6&9&9&6&3\\3&6&9&9&6&3\\2&4&6&6&4&2\\1&2&3&3&2&1\end{matrix}$$

Fig. A.1: Example of transposed deconvolution applied to a 4×4 matrix with a 3×3 kernel and a unitary stride.

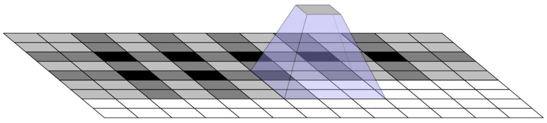


Fig. A.2: Visualisation of the checkerboard artefacts produced during the transposed convolution process. Source: *Odena et al.*, 2016[113].

Generative networks for near-field data prediction (see section 4.11) require upsampling of the input data to obtain the 4D output matrix from a 2D input matrix (with shape [*samples*×*parameters*]. In addition, both families of proposed architectures require upsampling to input the freestream scalars by concatenation with the data rakes matrix. This upsampling is achieved by the TensorFlow Conv2DTranspose layer, which implements the *transposed convolution*[58] operation, also improperly called *deconvolution*. An example of the transposed convolution operation is shown in the figure. The transposed convolution operation is known to produce checkerboard artefacts[113] that are difficult to avoid. For this reason, an architecture has also been proposed that does not use transposed convolution for upsampling but instead uses bilinear upsampling followed by a dimensionality-preserving convolution (i.e. the convolution output has the same shape as the input).

Dropout

As explained in section 2.5.2, the dropout layer was applied to the proposed neural networks to increase generalisation capacity and reduce overfitting. A design choice was made and, in line with the work that first introduced the dropout concept[56], Dropout layers were applied only after a fully connected layer; however, there are works in the literature that implement dropout layers after convolutional layers; albeit with lower dropout rates. The layers' dropout rate was then considered a hyperparameter for the HPO process. More information on dropout can be found in sections 2.5.2 and 4.12.

Tensor Flattening

Convolutional architectures for image classification and regression commonly flatten the feature maps obtained from the first portion of the neural network through convolution operations and then pass the flattened vector to one or more fully connected layers to obtain the network output. As described in section 4.11, airfoil performance prediction architectures follow this structure. The proposed models use, by default, the commonly used Flatten layer; however, during the HPO process, the effect of using the GlobalAveragePooling2D layer is investigated, which is currently employed in modern image classification architectures such as Mobilenets[114], ResNets[100], and EfficientNets[64].

While the Flatten Layer maintains the number of tensor elements and simply reshapes it to a vector, the GlobalAveragePooling2D layer applies the pooling operation. It averages all the values according to the last axis. That is, it preserves the size of the channel axis and batch axis, while the other spatial axes of the feature maps are reduced to 1.

Pooling

The pooling operations considered in this work during the HPO process are max pooling and average pooling. It should be noted that most modern CNN architectures do not use pooling layers, as they provide a reduction in tensor dimensionality without any learnable parameter and, therefore, cause a net loss of information within the network.

Batch normalization

Batch normalisation layers have been implemented to improve convergence speed. By shifting the input layers around the mean and scaling them by their variance[58], the network learning speed and training stability are improved[115–117]. In addition, batch normalisation could help with model performance and generalisation as it has been shown to have a weak regularisation effect and allows the use of larger learning rates[117].

Although it was initially claimed that this was due to a reduction in internal covariate shift, i.e. the change in distribution and variance of the layer inputs during training caused by updates in the upstream layers, the exact reasons for the effectiveness of the batch normalisation effect are not fully understood[116,117].

The layer works by normalising each mini-batch during training to produce an output with a mean of zero and a variance of 1[58]. Batch normalisation has been found to be more effective for larger mini-batch sizes, as the mini-batch statistics are found to be more representative of the entire dataset: the selected mini-batch size (see section 4.12.3) of 64 was found to be representative of the entire data set of 173 shuffled samples (see section 4.3).

Appendix

B TensorFlow Callbacks

Callbacks are a tool the TensorFlow API provides to customise model behaviour during the training and validation phases[58]. They allow the execution of custom code triggered by user-specified events that occur during training and validation, such as a call to a class method or the declaration of an attribute. The following callbacks were implemented in the models.

History The automatically applied History callback made it possible to obtain the model's loss and validation loss values for each epoch and batch.

Custom Early Stopping A custom early stopping callback (based on the standard EarlyStopping callback included in the TensorFlow API). It allows training to be stopped early if the validation loss of the model does not improve for a specific number of epochs (patience parameter, set to 200

epochs). In addition, when training is stopped, the trained model returned by the fit method is the one which corresponds to the lowest validation loss value obtained during training. This is accomplished by temporarily storing the model weights for a maximum number of epochs equal to the patience parameter.

CSV Logger It allowed the data of the epochs to be obtained in a numerical format and then stored in a CSV file to be used to plot the training curves of the models.

Reduce LR On Plateau The callback was used to dynamically adjust the learning rate of the optimizer (Adam) during training. It monitors a specified metric (the validation loss was the chosen metric) throughout the training process. If the chosen metric shows no improvement over a certain number of epochs (controlled by the patience parameter, set to 20), the learning rate is adjusted by a factor. This adaptive approach helps prevent the model from getting stuck in suboptimal solutions or converging too quickly.

TensorBoard TensorBoard is a tool provided by TensorFlow that enables the visualisation, debugging and testing of Keras and TensorFlow models. In particular, TensorBoard was used in this work to visualise the link between model performance and hyperparameter values during the hyperparameter optimisation process.

This callback provided the training process logs required to visualise the data via TensorBoard.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Stefano Berrone, for his support, advice, and guidance throughout the completion of this Master's thesis. His assistance was crucial to the completion of this thesis. I would also like to thank my co-supervisor, Prof. Gaetano Iuso, whose courses and lectures have sparked my intense fascination and curiosity for fluid dynamics, an interest I will retain regardless of my future career. I would like to express my gratitude to Francesco Della Santa and Fabio Vicini for their assistance and explanations, which enabled me to complete this thesis on a previously unfamiliar subject. Their input and constructive criticism were essential to the success of this project.

I would like to express my gratitude to the Politecnico di Torino for providing me with the opportunity to pursue a Master's degree and for the resources, facilities, and faculty support necessary to complete this thesis.

Additionally, I would like to thank the DAUIN department of the Politecnico di Torino and their HPC Section for providing the computing resources that made this work possible.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[2] J. Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *CoRR* abs/1404.7828 (2014).

[3] S. Sonoda and N. Murata. “Neural network with unbounded activation functions is universal approximator”. In: *Applied and Computational Harmonic Analysis* 43.2 (2017), pp. 233–268. ISSN: 1063-5203.

[4] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[5] J. N. Kutz. “Deep learning in fluid dynamics”. In: *Journal of Fluid Mechanics* 814 (2017), pp. 1–4.

[6] S. Suresh, S. Omkar, V. Mani, and T. G. Prakash. “Lift coefficient prediction at high angle of attack using recurrent neural network”. In: *Aerospace science and technology* 7.8 (2003), pp. 595–602.

[7] S. Ahmed, K. Kamal, T. A. H. Ratlamwala, S. Mathavan, G. Hussain, M. Alkahtani, et al. “Aerodynamic Analyses of Airfoils Using Machine Learning as an Alternative to RANS Simulation”. In: *Applied Sciences* 12.10 (2022), p. 5194.

[8] F. G. Oztiryaki and T. Piskin. “Airfoil Performance Analysis Using Shallow Neural Networks”. In: *AIAA Scitech 2021 Forum.*

[9] M. Morimoto, K. Fukami, K. Zhang, A. G. Nair, and K. Fukagata. “Convolutional neural networks for fluid flow analysis: toward effective metamodeling and low dimensionalization”. In: *Theoretical and Computational Fluid Dynamics* 35.5 (Aug. 2021), pp. 633–658.

[10] Q. Wang, C. E. Cesnik, and K. Fidkowski. “Multivariate recurrent neural network models for scalar and distribution predictions in unsteady aerodynamics”. In: *AIAA Scitech 2020 Forum.* 2020, p. 1533.

[11] H. Chen, L. He, W. Qian, and S. Wang. “Multiple aerodynamic coefficient prediction of airfoils using a convolutional neural network”. In: *Symmetry* 12.4 (2020), p. 544.

[12] B. Yu, L. Xie, and F. Wang. “An Improved Deep Convolutional Neural Network to Predict Airfoil Lift Coefficient”. In: Jan. 2020, pp. 275–286. ISBN: 978-981-15-1772-3.

[13] J. Ling, A. Kurzawski, and J. Templeton. “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance”. In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166.

[14] Y. Frey Marioni, E. A. de Toledo Ortiz, A. Cassinelli, F. Montomoli, P. Adami, and R. Vazquez. “A machine learning approach to improve turbulence modelling from DNS data using neural networks”. In: *International Journal of Turbomachinery, Propulsion and Power* 6.2 (2021), p. 17.

[15] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. “Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows”. In: *AIAA Journal* 58.1 (2020), pp. 25–36.

[16] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik. “Prediction of aerodynamic flow fields using convolutional neural networks”. In: *Computational Mechanics* 64 (2019), pp. 525–545.

[17] J. Holland, J. Baeder, and K. Duraisamy. “Towards Integrated Field Inversion and Machine Learning With Embedded Neural Networks for RANS Modeling”. In: Jan. 2019.

[18] C. Schenck and D. Fox. “SPNets: Differentiable Fluid Dynamics for Deep Neural Networks”. In: *CoRR* abs/1806.06094 (2018).

[19] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. “NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 426 (2021), p. 109951.

[20] Q. Zhu, Z. Liu, and J. Yan. “Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks”. In: *Computational Mechanics* 67 (2021), pp. 619–635.

[21] S. Feng, H. Zhou, and H. Dong. “Using deep neural network with small dataset to predict material defects”. In: *Materials & Design* 162 (2019), pp. 300–310.

[22] P. Cao, S. Zhang, and J. Tang. “Preprocessing-free gear fault diagnosis using small datasets with deep convolutional neural network-based transfer learning”. In: *Ieee Access* 6 (2018), pp. 26241–26253.

[23] T. Shaikhina and N. A. Khovanova. “Handling limited datasets with neural networks in medical applications: A small-data approach”. In: *Artificial intelligence in medicine* 75 (2017), pp. 51–63.

[24] A. Pasini. “Artificial neural networks for small dataset analysis”. In: *Journal of thoracic disease* 7.5 (2015), p. 953.

[25] J. Bullock, C. Cuesta-Lázaro, and A. Quera-Bofarull. “XNet: A convolutional neural network (CNN) implementation for medical X-Ray image segmentation suitable for small datasets”. In: *CoRR* abs/1812.00548 (2018).

[26] L. Brigato and L. Iocchi. “A Close Look at Deep Learning with Small Data”. In: *CoRR* abs/2003.12843 (2020). URL: https://arxiv.org/abs/2003.12843.

[27] S. Liu and W. Deng. “Very deep convolutional neural network based image classification using small training sample size”. In: *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE. 2015, pp. 730–734.

[28] K. Choi, G. Fazekas, M. B. Sandler, and K. Cho. “Transfer learning for music classification and regression tasks”. In: *CoRR* abs/1703.09179 (2017). URL: http://arxiv.org/abs/1703.09179.

[29] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. “A survey on deep transfer learning”. In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*. Springer. 2018, pp. 270–279.

[30] D. C. Wilcox. *Turbulence Modeling for CFD*. third. DCW Industries, 2006. ISBN: 978-1-928729-08-2.

[31] G. Buresti. “A note on stokes’ hypothesis”. In: *Acta Mechanica* 226.10 (2015), pp. 3555–3559.

[32] M. V. Papalexandris. “On the applicability of stokes’ hypothesis to low-mach-number flows”. In: *Continuum Mechanics and Thermodynamics* 32.4 (2019), pp. 1245–1249.

[33] J. B. J. Fourier and A. Freeman. *The analytical theory of heat*. New York: Dover Publications, Inc, 1955.

[34] Y. Park and R. E. Sonntag. “Thermodynamic properties of ideal gas air”. In: *International Journal of Energy Research* 20.9 (1996), pp. 771–785.

[35] J. D. Anderson. *Modern Compressible Flow: With Historical Perspective*. Mc Graw Hill Education, 2020. ISBN: 978-1260471441.

[36] J. A. Dean. *Lange’s handbook of chemistry*. Mc Graw Hill Education, 1999. ISBN: 0-07-016384-7.

[37] W. Sutherland. “The viscosity of gases and molecular force”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 36.223 (1893), pp. 507–531.

[38] S. B. Pope. *Turbolent flows*. Mc Graw Hill Education, 2017.

[39] NASA. *nasa.gov*. 2023. URL: https://www.nasa.gov (visited on 03/10/2023).

[40] J. D. Anderson. *Fundamentals of aerodynamics*. Cambridge University Press, 2017.

[41] E. Buckingham. “On Physically Similar Systems; Illustrations of the Use of Dimensional Equations”. In: *Phys. Rev.* 4 (4 1914), pp. 345–376.

[42] C. P. Coutinho, A. J. Baptista, and D. R. José. “Reduced scale models based on similitude theory: A review up to 2015”. In: *Engineering Structures* 119 (2016), pp. 81–94.

[43] R. Arina. *Fondamenti di aerodinamica*. Levrotto & Bella, 2015. ISBN: 8-88-218187-1.

[44] D. Tritton. *Physical FLuid Dynamics*. First. Oxford University Press, 1988. ISBN: 978-0198544937.

[45] E. N. Lorenz. “Deterministic Nonperiodic Flow”. In: *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–141.

[46] L. F. Richardson. *Weather prediction by numerical process*. University Press, 1922.

[47] J. A. S. and Rodney D. W. Bowersox. *Boundary Layer Analysis, Second Edition*. Second. American Institute of Aeronautics and Astronautics, 2011. ISBN: 978-1-60086-823-8.

[48] ANSYS Inc. *Ansys Fluent Theory Guide*. Version Release 2022 R2. July 2022.

[49] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613.

[50] W. Mcculloch and W. Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[51] A. Ivakhnenko and V. Lapa. *Cybernetic Predicting Devices*. Jprs report. CCM Information Corporation, 1973.

[52] S. Haykin. *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall, 1999. ISBN: 9780132733502.

[53] J. Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080.

[54] S. Linnainmaa. “Taylor Expansion of the Accumulated Rounding Error”. In: *BIT* 16.2 (1976), pp. 146–160.

[55] MathWorks, Inc. 2023. URL: https://www.mathworks.com/discovery/overfitting.html (visited on 01/16/2023).

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[57] K. Pasupa and W. Sunhem. “A comparison between shallow and deep architecture classifiers on small dataset”. In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. 2016, pp. 1–6.

[58] Google LLC. *Tensorflow Python API Documentation*. 2023. URL: https://www.tensorflow.org/api_docs/python/ (visited on 01/16/2023).

[59] K. Fukushima. “Cognitron: A self-organizing multilayered neural network”. In: *Biological cybernetics* 20.3 (1975), pp. 121–136.

[60] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *CoRR* abs/1811.03378 (2018).

[61] T. Szandała. “Review and comparison of commonly used activation functions for deep neural networks”. In: *Bio-inspired neurocomputing* (2021), pp. 203–224.

[62] S. Hayou, A. Doucet, and J. Rousseau. “On the impact of the activation function on deep neural networks training”. In: *International conference on machine learning*. PMLR. 2019, pp. 2672–2680.

[63] P. Ramachandran, B. Zoph, and Q. V. Le. *Searching for Activation Functions*. 2017.

[64] M. Tan and Q. V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *CoRR* abs/1905.11946 (2019). URL: http://arxiv.org/abs/1905.11946.

[65] D. Hendrycks and K. Gimpel. “Gaussian Error Linear Units (GELUs)”. In: (2016).

[66] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. “Improving language understanding by generative pre-training”. In: (2018).

[67] X. Zhang, D. Chang, W. Qi, and Z. Zhan. “A Study on Different Functionalities and Performances among Different Activation Functions across Different ANNs for Image Classification”. In: *Journal of Physics: Conference Series* 1732.1 (Jan. 2021), p. 012026.

[68] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, et al. “Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions”. In: *Journal of big Data* 8 (2021), pp. 1–74.

[69] *ImageNet Large Scale Visual Recognition Challenge*. URL: https://www.image-net.org/challenges/LSVRC/ (visited on 02/05/2023).

[70] C. D. Harris. *NASA Supercritical Airfoils: A Matrix of Family-Related Airfoils*. Tech. rep. NASA, 1990.

[71] HPC@POLITO. *HPC@POLITO*. URL: https://www.hpc.polito.it/about.php (visited on 12/12/2022).

[72] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[73] R. Eymard, G. Thierry, and R. Herbin. “Finite Volume Methods”. In: 7 (Dec. 2000).

[74] *Airfoil Tools.* URL: http://airfoiltools.com/ (visited on 02/05/2023).

[75] J. Thompson, B. Soni, and N. Weatherill. *Handbook of Grid Generation.* CRC Press, 1998. ISBN: 9781420050349.

[76] Langley Research Center. *Turbulence Modeling Resource.* 2021. URL: http://turbmodels.larc.nasa.gov/naca0012_val.html (visited on 01/13/2023).

[77] ANSYS Inc. *Ansys Fluent Users Guide.* Version Release 2022 R2. July 2022.

[78] B. S. Stratford, G. S. Beavers, and The Aeronautical Research Council and National Gas Turbine Establishment. *The Calculation of the Compressible Turbulent Boundary Layer in an Arbitrary Pressure Gradient: A Correlation of Certain Previous Methods.* ARC-21399. H.M. Stationery Office, 1959.

[79] H. Schlichting, C. Mayes, E. Krause, H. J. Oertel, and K. Gersten. *Boundary-Layer Theory.* Springer Berlin Heidelberg, 2003. ISBN: 9783540662709.

[80] N. T. Ouellette. “Turbulence in two dimensions”. In: *Physics Today* 65.5 (2012), pp. 68–69.

[81] M. V. Morkovin. “Effects of compressibility on turbulent flows”. In: *Mécanique de la Turbulence* 367.380 (1962), p. 26.

[82] F. R. Menter. “Two-equation eddy-viscosity turbulence models for engineering applications”. In: *AIAA Journal* 32.8 (1994), pp. 1598–1605.

[83] J. Boussinesq. *Theorie Analytique de La Chaleur Mise En Harmonie Avec La Thermodynamique Et Avec La Theorie Mecanique de la Lumiere, Tome II.* Gauthier-Villars, 1903.

[84] A. N. Kolmogorov. “Equations of turbulent motion in an incompressible fluid”. In: *Dokl. Akad. Nauk SSSR* 30.4 (1941), pp. 299–303.

[85] D. C. Wilcox. “Formulation of the k-w Turbulence Model Revisited”. In: *AIAA Journal* 46.11 (2008), pp. 2823–2838.

[86] F. Menter, R. Langtry, S. Likki, Y. Suzen, P. Huang, and S. Völker. “A Correlation-Based Transition Model Using Local Variables—Part I: Model Formulation”. In: *ASME J. Turbomach* 128 (July 2006).

[87] P. Malan, K. Suluksna, and E. Juntasaro. “Calibrating the y-Reθ transition model for commercial CFD”. In: *AIAA* (2009), p. 1142.

[88] B. Kader. “Temperature and concentration profiles in fully turbulent boundary layers”. In: *International Journal of Heat and Mass Transfer* 24.9 (1981), pp. 1541–1544. ISSN: 0017-9310.

[89] C. Britcher and D. Landman. *Wind Tunnel Test Techniques: Design and Use at Low and High Speeds with Statistical Engineering Applications*. Elsevier Science, 2023. ISBN: 9780128180990.

[90] M.-S. Liou. “A sequel to ausm: Ausm+”. In: *Journal of computational Physics* 129.2 (1996), pp. 364–382.

[91] S.-s. Kim, C. Kim, O.-H. Rho, and S. K. Hong. “Cures for the shock instability: development of a shock-stable Roe scheme”. In: *Journal of Computational Physics* 185.2 (2003), pp. 342–374.

[92] S. Phongthanapanich and K. Takayama. “A Comparison of the Roe’s FDS, HLLC, AUFS, and AUSMDV+ Schemes on Triangular Grids”. In: *Applied Science and Engineering Progress* 12.3 (2019), pp. 150–157.

[93] A. Pope and K. Goin. *High Speed Wind Tunnel Testing*. John Wiley & Sons, 1965. ISBN: 9780471694021.

[94] R. Jenkins. *NASA SC(2)-0714 Airfoil Data Corrected for Sidewall Boundary-layer Effects in the Langley 0.3-meter Transonic Cryogenic Tunnel*. NASA technical paper. NASA, Scientific and Technical Information Division, 1989. URL: https://books.google.it/books?id=6qUTAQAAIAAJ.

[95] R. Bartels and J. Edwards. “Cryogenic Tunnel Pressure Measurements on a Supercritical Airfoil for Several Shock Buffet Conditions”. In: (Oct. 1997).

[96] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org.

[97] J. Sola and J. Sevilla. “Importance of input data normalization for the application of neural networks to complex industrial problems”. In: *IEEE Transactions on nuclear science* 44.3 (1997), pp. 1464–1468.

[98] M. M. Bejani and M. Ghatee. “A systematic review on overfitting control in shallow and deep neural networks”. In: *Artificial Intelligence Review* (2021), pp. 1–48.

[99] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. “ coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[100] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[101] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. *Densely Connected Convolutional Networks*. 2018.

[102] H. Alshazly, C. Linse, M. Abul-Dahab, E. Barth, and T. Martinetz. “COVID-Nets: deep CNN architectures for detecting COVID-19 using chest CT scans”. In: *PeerJ Computer Science* 7 (July 2021), e655.

[103] T. O’Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. *Keras-Tuner*. https://github.com/keras-team/keras-tuner. 2019. (Visited on 01/16/2023).

[104] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52.

[105] Google LLC. *TensorBoard*. https://github.com/tensorflow/tensorboard. 2023. (Visited on 01/16/2023).

[106] O. A. Olayemia, O. I. Salakoa, A. Jinadu, and A. Martins. “Aerodynamic lift coefficient prediction of supercritical airfoils at transonic flow regime using convolutional neural networks (CNNs) and multi-layer perceptions (MLPs)”. In: *Al-Qadisiyah Journal for Engineering Sciences* 16.2 (2023).

[107] A. Arias-Montano, C. Coello, and E. Mezura-Montes. “Evolutionary Algorithms Applied to Multi-Objective Aerodynamic Shape Optimization”. In: *Studies in Computational Intelligence* 356 (2011), pp. 211–240.

[108] R. Derksen and T. Rogalsky. “Bezier-PARSEC: An optimized aerofoil parameterization for design”. In: *Advances in engineering software* 41.7-8 (2010), pp. 923–930.

[109] M. Khurana, H. Winarto, and A. Sinha. “Application of Swarm Approach and Artificial Neural Networks for Airfoil Shape Optimization”. In: *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, MAO* (Sept. 2008).

[110] H. Chen, L. He, W. Qian, and S. Wang. “Multiple Aerodynamic Coefficient Prediction of Airfoils Using a Convolutional Neural Network”. In: *Symmetry* 12 (Apr. 2020), p. 544.

[111] Y. Zhang, W. Sung, and D. N. Mavris. “Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient”. In: *ArXiv* abs/1712.10082 (2017). URL: https://api.semanticscholar.org/CorpusID:19039547.

[112] ANSYS Inc. *Ansys Fluent Text Command List*. Version Release 2022 R2. July 2022.

[113] A. Odena, V. Dumoulin, and C. Olah. “Deconvolution and checkerboard artifacts”. In: *Distill* 1.10 (2016), e3.

[114] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[115] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[116] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. “How does batch normalization help optimization?” In: *Advances in neural information processing systems* 31 (2018).

[117] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. “Understanding batch normalization”. In: *Advances in neural information processing systems* 31 (2018).