POLITECNICO DI TORINO

Department of Mechanical Engineering

Master's Degree Thesis

(2020-2021)

Prediction of the laminar combustion velocity in methane-air mixtures by means of deep learning algorithms

Thesis advisors

Daniela Anna Misul

Mirko Baratta

Candidate

Davide Fittipaldi

Abstract

Prediction of the laminar combustion velocity in methane-air mixtures by means of deep learning algorithms

The need to reduce environmental pollution has led to a search for alternative, low-emission fuels. In fact, internal combustion engines, are one of the main source of pollution in the urban environment. Methane is one of the best alternatives to the main and most common fossil fuels, because it is one of the fuels with the lowest CO_2 and hydrocarbon emissions. In addition to being one of the cleanest fuels, it has remarkable chemical and physical properties, such as high anti-knocking, which would allow high efficiency in monovalent methane-fuelled heat engines with an high compression ratio. However, compared to petrol, methane burns slowly, which leads to a variation in efficiency and not complete stability cycle by cycle, reducing power and increasing fuel consumption. Low flame front propagation speed and poor burning capacity in poor mixture conditions can be improved by the addition of hydrogen, due to its higher burning speed.

The aim of this thesis is to develop a deep learning algorithms for predicting the laminar speed of combustion in methane/air mixtures, providing as input the conditions of the mixture (pressure, temperature, percentage of EGR, equivalent ratio). The objective of this methodology is to see if it is possible to construct a neural network capable of understanding the non-linear relationships of the phenomenon under analysis, allowing a faster simulation, by virtue of the reduction of computational costs, but at the same time must be accurate, respecting what is the physics and kinetics of the combustion process. The results obtained by training our neural network separately on the combustion tables obtained with the main chemical kinetics mechanisms, i.e. Aramco 2.0 and GRImech 3.0, were also compared with the laminar flame speed experimental data available in the literature.

Another important objective of this thesis is to continue research in the field of heat engines, in particular to generate combustion tables using LOGEresearch, software for simulating the chemical kinetics of the combustion process, for air/methane mixtures with and without the addition of hydrogen. The work focuses on finding the mixture dosage, with a value that differs from one, to observe how the value of the laminar speed varies, compared to the other input values, which are used to start the simulation.

Acknowledgements

To my family, who have made all this possible, especially my parents, my brother and all those who have supported me. Thanks to the friends I have met over the years, and thanks to my lifelong friends who have always supported me.

Contents

List of figures

1 Introduction

2 Introduction on Machine Learning and Artificial Neural Networks

3 First results

3.1 First attempt Aramco 2.0 results

3.2 First attempt Gri-mech 3.0 results

3.3 Methods for hyperparameter optimisation

3.3.1 Grid search optimised hyperparameters: results with Aramco 2.0 dataset

3.3.2 Grid search optimised hyperparameters: results with Gri-mech 3.0 dataset

3.4 Grid search with Cross Validation

3.4.1 Grid Search with Cross Validation: results with Aramco 2.0 dataset

3.4.2 Grid search with Cross Validation: results with Gri-mech 3.0 dataset

List of tables

2 Introduction on Machine Learning and Artificial Neural Networks

1 Introduction

The requirements of European emission standards, which define the maximum acceptable limit for new vehicles, influence the development and design of new technologies. The development of alternative petroleum fuels in internal combustion engines has been necessary, such as natural gas (CNG), which is mainly composed of methane, has a low carbon content and therefore a lower level of emissions in terms of carbon dioxide, carbon monoxide and hydrocarbon. As far as nitrogen oxides are concerned, they will decrease due to the lower temperature reached in the cylinder. However, nowadays there is a tendency to develop ever smaller and turbocharged powertrains, in the case of the latter, the temperature inside the cylinder will definitely increase resulting in higher NOx emissions. Therefore, an EGR system will be used to prevent the formation of nitrogen oxides because the exhaust gases can act as diluents in the unburned gas mixture, and the peak temperature reached during the combustion process decreases as the residual concentration increases. The amount of recirculated exhaust gas must be monitored because it produces a less reactive mixture with possible errors or ignition problems. In fact, at the end of the combustion process, the pressure peak and the flame front, during the expansion phase, usually slow down due to the lower pressure and temperature of the mixture. If the temperature drops too quickly due to excessive EGR opening, the flame front can stop, leaving a portion of the mixture unburned. It should be remembered that natural gas has an antiknock property that enables a spark-ignition engine to be exploited, using a higher compression ratio, and consequently increasing thermal efficiency. On the other hand, methane burns slowly, adversely affecting the efficiency, available power and fuel consumption of the engine due to variations in the combustion process, which is not constant from cycle to cycle. Consequently, blending hydrogen with natural gas can help address these problems due to its reactivity, ensuring improved thermal efficiency, increasing combustion speed, extending flammability limits and reducing pollutant emissions due to a higher atomic hydrogen to carbon ratio. For these reasons, an accurate calculation of laminar combustion velocities is necessary for the design of spark ignition engines.

Spark ignition engines (S.I., Otto, gasoline engine) can use fuels with relatively high ignition delays, i.e. low reactivity, such as petrol, methanol, ethanol, natural gas, LPG. The fuel, which can be compressed without giving rise to combustion reactions, is pre-mixed with combustion air generally outside the cylinder. The combustion process is initiated from the outside by an electric spark, which is able to bring a small portion of the mixture to a temperature of over 1000 K, and from this initial ignition nucleus combustion spreads to the rest of the charge under conditions of a turbulent regime (combustion in turbulent air). An initial nucleus of combustion gases is formed which transmits heat by means of a heat exchange mechanism to the small layer of adjacent mixture, raising its temperature, causing oxidation reactions to take place in this layer as well, with the subsequent release of thermal energy, and subsequent propagation of the flame front until it reaches the areas furthest from the spark plug. This combustion process takes place in the gaseous phase in a "preformed" mixture, so that the process of mixture formation and combustion take place at two separate times, and only once the mixture is homogeneous can combustion take place. Combustion consists of a series of chemical reactions in which a fuel is oxidised by an oxidiser and heat is generated, electromagnetic and even light radiation. Combustion is very often also accompanied by the presence of a flame and the production of high-temperature gases produced by combustion, dispersing dust (usually carbonaceous soot), obtained from combustion and giving rise to smoke. In other words, combustion is an exothermic oxidation-reduction reaction, in which one compound is oxidised while another is reduced, in hydrocarbons, carbon is oxidised while oxygen is reduced, releasing energy and forming water and carbon dioxide. We can classify and identify different combustion processes, in fact the reactions between fuel and oxidiser can take place[13]:

- Without flame: the reaction is very slow and takes place at temperatures around 500-600°C, low temperatures and with a response time up to ten seconds.

- With deflagrating flames: in this case most of the reactions take place within the thickness of the flame, with reaction times of less than a millisecond, and a flame front velocity of 20-30 m/s. During this type of combustion there is a drop in density and pressure in the area occupied by the products of combustion, which causes an increase in the speed of propagation of the front itself, which is also driven by the expansion of the combustion gases. This drop in pressure is a negligible fraction of the average pressure in the domain in which combustion takes place. This pressure drop is a negligible fraction of the average pressure in the domain in which combustion takes place. isobaric process, i.e. pressure variations across the domain are negligible compared to those of temperature or kinetic energy.
- With detonating flames: in this case, the propagation speeds of the flame front become particularly high, with peaks reaching values of 1000 m/s, with the consequent formation of shock waves. This phenomenon occurs when there is simultaneous self-ignition of the entire mass of the mixture present, which does not require propagation but detonation of the entire mass present in the domain under consideration. Comparing deflagration with detonation, in the latter case there is an increase in pressure and density, and therefore a deceleration of the flame front, unlike deflagration. In the case of deflagration, we also have a different pressure trend, we can assume that there is no pressure gradient, while in the case of detonation there will not only be the typical step trend of the temperature profile as it crosses the flame front, but there will also be a pressure step [13].

Another possible classification of flames can be made by distinguishing them into laminar and turbulent flames:

- Laminar flames: are characterised by having a well-defined flame front with a regular surface. The speed of propagation of the flame front is particularly low and the thickness of the laminar flame front is between 0.05-0.20 mm.
- Turbulent flames: in this type of flame, the flame area is still identified but the flame front is wrinkled and irregular. The increase in the surface area of the flame front due to turbulent motion leads to an increase in the flame front propagation velocity which increases as the intensity of the turbulence increases. The thickness of the turbulent flame front is between 3 and 9 mm, so it is no longer negligible as in the case of laminar flame front thickness [13].

Laminar and turbulent flames can be classified in turn into:

- Diffusive flames: in this type of flame the oxidiser and the fuel are introduced separately. Therefore, before the chemical reactions between the reactants can begin, the fuel must be in a gaseous state. Therefore, before the chemical reactions between the reactants can begin, the fuel must be in a gaseous state and there must also be mixing between the various reactants. A practical example of this type of flame is the candle flame, while a practical example is combustion in compression ignition engines [13].
- Pre-mixed flames: in this type of flame, combustion takes place after mixing the fuel and the oxidiser. A practical example of this type of flame is the home cooker, while a technical example is the combustion of petrol or methane in spark ignition engines [13].

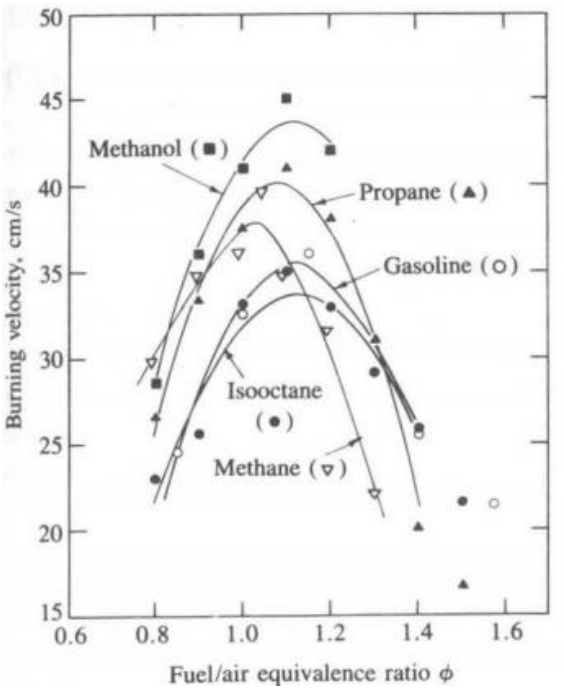

Figure 1: Laminar propagation velocity trend as the equivalence ratio changes, for different fuel types, for T = 300 K and p = 1 bar [14]

In figure 1 we can see the trend of the laminar speed of propagation which is sensitive to the variation of the composition of the mixture. We must underline that the combustion process could not take place with the characteristic speeds of a laminar flame, which is two orders of magnitude less than the speeds necessary for the process. The speeds would be insufficient given the short time available, even with a stoichiometric ratio that gives the highest laminar flame speeds. From the graph, we see how the laminar speed varies as the equivalence ratio (Ø) varies, and we note that the speeds are different according to the composition of the mixture, as we expected, however there are other factors that affect the speed such as T, p and X_{burn} (which takes into account the presence of diluents such as EGR which slow down the flame front).

The graph represented by the figure 1, follows the following equation:

$$v_L = v_{L0} \cdot \left(\frac{T_X}{T_0}\right)^2 \cdot \left(\frac{P_X}{P_0}\right)^{-0.25} \cdot (1 - cx_{burn})^{0.7}$$

With v_{L0} taken from the graph and referred to T_0 = 300 K and p_0 = 1 bar. At the same X_{burn}, with T_x = 700 K and p_x = 2 bar, v_L roughly doubles with the same X_{burn}.

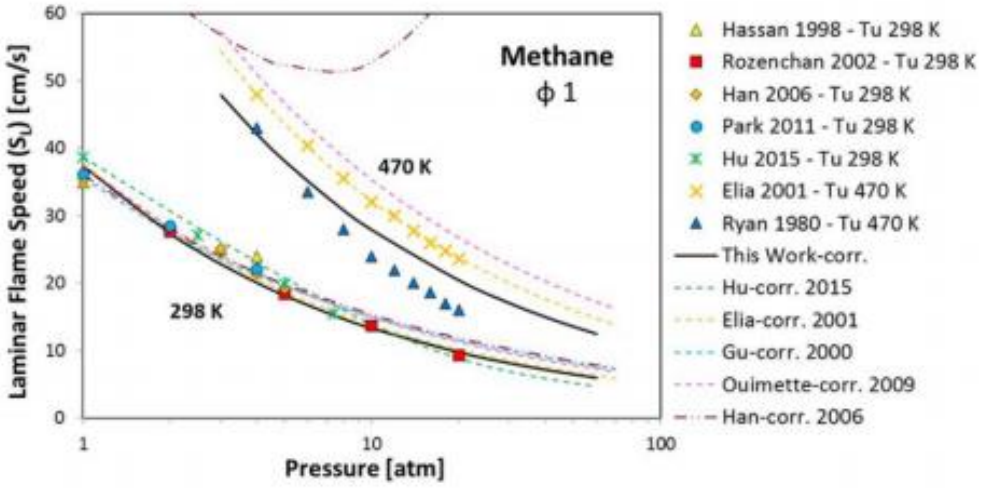

Figure 2: Experimental LFS trends at varying pressure, for a stoichiometric mixture of methane at 298 K and 470 [15]

In order to understand the inverse dependence of flame front speed on pressure, i.e. with a denser mixture, it is necessary to analyse the combustion phenomenon. The process involves a convective heat exchange between what has already been burned, combusted gas, and the layer of unburned gas. In this case there is a heat flow from the burned gas to the unburned gas, the latter having a higher density and therefore a larger mass, which requires a greater heat exchange from the burned gas to raise the pressure and temperature necessary for the combustion process. In addition, a higher temperature results in a smaller temperature gradient between the charge and the burned gas, but as a consequence the charge density is lower and this effect prevails, so less heat is required to raise the temperature of the unburned gas. The further correction factor for the presence of diluents is $(1 - cx_{burn})^{0.7}$, and where inert diluents are present absorb heat, so the only effect they have is to reduce or slow down the propagation rate.

As mentioned above, the turbulent front speed is greater than the laminar front speed; it is the turbulence that increases the flame front speed. This is because it is the motions inside the chamber that make the surface jagged, with a multiplication effect of the surface that assuming the form of a thin foil corrugated and folded on itself several times with a fractal-like geometry, which allows us to recover those two orders of magnitude that were missing. According to an early model proposed by Damkohler (1940), this increase, essentially attributable to the increase in the separation surface between the and fresh charge, can be expressed as:

$$w_T = w_L \, (A_T/A_L)$$

Where:

- A_T: is the effective corrugate area, in turbulent condition, that is increased than A_L
- A_L: is the plane combustion area characterize from a laminar flame

Furthermore, according to Damkohler, $(A_T/A_L) \sim u'$ where u' is the turbulent intensity, directly proportional to the mean speed. The combustion process adapts to the operating conditions present in a given situation.

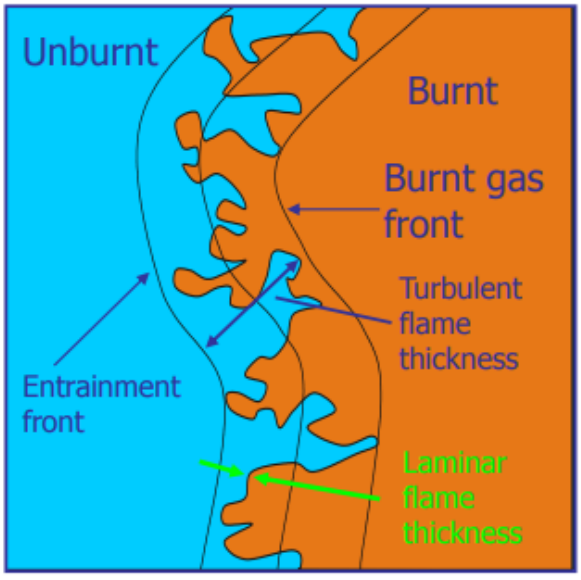

Figure 3: Turbulence effect on corrugation and thickness of the flame front [14]

1.1 Review of chemical kinetics and laminar flame speed dependence

During deflagrating combustion, as previously reported, the pressure remains uniform throughout the region concerned, and during this combustion there is a decrease in density on the part of the combustion products, which causes an increase in the propagation speed of the flame front, a result achieved by counting the transport component and the front component. The laminar flame front propagation speed is one of the most important properties in the study of combustion because many phenomena depend on this quantity, such as the structure and speed of turbulent flames, flame front instability, front stretching, and it plays a key role in many technical applications involving combustion such as internal combustion engines, gas turbines, burners, and explosion prediction. In the case of internal combustion engines, the value of the laminar velocity of front propagation influences [14]:

- The value of the *spark advance*, and consequently the engine's performance and emissions.
- *Cyclic dispersion*, a phenomenon whereby a cycle, even considering the same cylinder, tends to be different from the previous one.
- *Wall quench layers*, an important factor in the production of hydrocarbons. This is the thickness within which combustion does not continue due to the low temperature of the combustion chamber walls.

The laminar velocity is an intrinsic characteristic of a fuel and the main parameters that influence it are temperature, pressure, equivalence ratio (ϕ), percentage of combustion gases in the mixture. The parameter that most influences the Laminar Flame Speed (LFS) is temperature, because at the base of the flame front propagation there is a large number of parameters. This is because there are chemical reactions at the base of the flame front propagation. The faster these reactions are completed, the more the LFS increases. To better understand this aspect, it is necessary to remember some concepts of chemical kinetics, a fundamental physical phenomenon for physical events of such short duration. The combustion of a generic hydrocarbon C_xH_y, even in the gaseous phase and pre-mixed homogeneously with the combustion air, takes place according to a multi-stage process, with the formation of intermediate reaction products (radicals, peroxides, etc.). The explosion mechanism leading to ignition of the mixture is due to chain reactions producing large quantities of radicals. Ignition of the mixture therefore only occurs after a certain ignition delay. The combustion of a hydrocarbon therefore involves the oxidation of the fuel with a series of intermediate reactions. Hydrogen atoms are the first to break their bond with carbon, and then bind with oxygen, resulting in the formation of a series of intermediate compounds. There are several ways in which this phenomenon can occur as shown in the figure below.

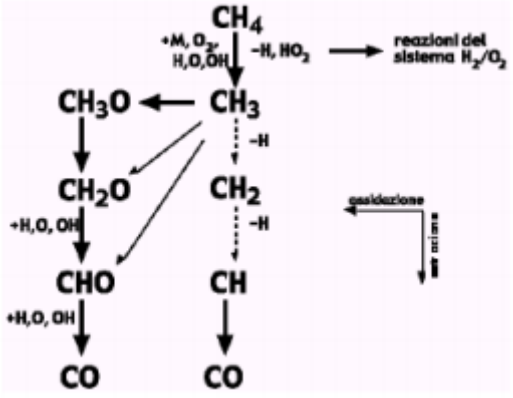

Figure 1.1.1: Pre-reaction involving the methane molecule [14]

To describe the combustion process in detail, even of a single pure hydrocarbon, requires high computational efforts. We can therefore consider, the common factor in these processes, where in general, the release of

major chemical energy occurs in the last steps of the process. The first reactions are essential for the formation of the intermediate compounds, which are so unstable, and this instability increases as the last steps of the reaction become almost instantaneous. This is because during the development of the reaction, there is an increase in the concentration of the radicals (intermediate compounds), which lead to the sudden development of the exothermic reaction. A certain amount of time, called the ignition delay, elapses before the exothermic reaction takes place. The speed at which intermediate reactions proceed can be assessed using relation (1), given a generic combustion reaction, the speed at which the reactants are consumed (or at which the reaction proceeds from left to right) is given by the following relation for forward reaction:

$$R^+ = k^+ \prod_{i=1}^{n} [M_{Ri}]^{\nu_{Ri}} \quad (1)$$

Where the constant k^+ follows Arrhenius' law:

$$k^+ = a\, T^b e^{-\frac{E_a}{RT}} \quad (2)$$

Where E_a is the activation energy of a generic reaction, and with '*a*' and '*b*' depend on the type of the reaction. Then we see from the equation (2) that the speed with which a product of a generic reaction is generated, depends exponentially on the temperature [14]. The exponential coefficient, also known as the Boltzman factor, defines the fraction of collisions that have an energy greater than the activation energy and allow a reaction to take place. In other words, the chemical reaction takes place once the collision between the molecules has released enough kinetic energy for the bonds to dissolve and allow others to form. So this coefficient K^+ expresses the probability that two molecules have of colliding and breaking bonds. The higher the temperature, the greater this probability. The lower is the activation energy, the greater is the probability.

We have seen the effects of temperature and pressure on the laminar speed, we can also identify another fundamental parameter, that is the stoichiometric dosage defined as the correct air mass to fuel ratio that perfectly completes the chemical reaction of fuel oxidation, without excess of oxidant or fuel. It can be determined considering the ideal reaction of oxidation of a generic non-oxygenated fuel that leads to have a ratio $(air/fuel)_{st}$ depends only on the ratio between hydrogen atoms and oxygen atoms (y/x) that in the case of gasoline or diesel, this ratio is equal to about 1.85, leading to a stoichiometric dosage equal to 14,6 while in the case of methane (y/x) is equal to 4 and then the air fuel stoichiometric ratio is equal to 17,23. Regarding the dependence of the LFS on this parameter, we will have the maximum propagation velocity in the stoichiometric range, or rather in the slightly rich range, and then collapse for values far from the stoichiometric. The reason why the peak of the speed of propagation of the flame front is in the slightly rich range is to be found in the adiabatic temperature of the flame, which will have a peak in the slightly rich range, causing an acceleration of the reactions, having an exponential dependence on the temperature and therefore on the speed of propagation of the front. The adiabatic flame temperature has a peak in the slightly rich because in this zone there is more production of carbon monoxide than carbon dioxide, causing a decrease in the specific heat of the products, because triatomic gases absorb more energy than diatomic ones, consequently less energy will be absorbed by the products and this will increase the adiabatic flame temperature. This reasoning is restricted only to this area of the equivalence ratio, as for values of equivalence, since for values far from the stoichiometric one will have either an excess of air or of fuel, and since everything that does not burn absorbs energy, there will be a decrease in the adiabatic flame temperature and therefore in the speed of propagation of the front. As can be seen from Figure 1.1, the LFS curves are only represented in a portion of the equivalence ratio, this portion being bounded by the flammability limits. In fact, two flammability limits are defined:

- Upper flammability limit: define as maximum percentage by volume of fuel in air for which there is propagation of the flame front; in fact, when there is an excess of fuel, the reaction instead of being endothermic becomes exothermic causing the flame front to stop.
- Lower flammability limit: define as minimum percentage by volume of fuel in air for which there is flame propagation.

We must emphasise the importance of adding hydrogen as an "additive" for the combustion of methane, hydrogen has a considerable flammability range which is able to burn in conditions which are also very far from stoichiometric conditions, this, together with the very high speed of propagation of the flame front, represents one of the points of strength of the fuel. This, together with the very high speed of flame propagation, is one of the strengths of this fuel. This is why alternative fuels such as methane mixed with hydrogen are being studied in order to combine the strengths of two different fuels.

Another very important factor is the dependence on the fraction of combustion gases in the mixture. Very often in motorsport, the reuse of combustion gases is used to reintroduce them into the combustion chamber in order to reduce emissions or to de-throttle partial loads in the case of spark ignition engines. This technique is known as EGR (Exhaust Gas Recirculation). The impact of this fraction of combustion gases on the LFS is negative, causing it to slow down for various reasons, firstly by reducing the temperatures in the chamber, the speed of the reactions will also be reduced and therefore the speed of propagation, but it will also affect the concentrations of the reactants, making it less likely that the various reactants will collide.

1.2 Laminar flame front structure

The structure of the laminar flame front can be divided into three zones [13]:

- In the first part, the temperature rises exponentially, while the enthalpy of formation remains constant, which means that combustion has not yet begun. However, the first reactions take place which lead to the dehydrogenation of the fuel molecule by radicals coming from the reaction zone by diffusion. The controlling phenomenon at this stage is therefore the diffusion of the radicals.
- In the second zone, the increase in temperature and enthalpy of formation becomes linear, this zone is called the flame thickness. In this zone diffusion transport can be considered negligible compared to convection transport. At the end of this zone, practically all the fuel will have been oxidised.
- In the third region, both the enthalpy of formation and the temperature increase very slowly. This zone is called the post-combustion zone.

In order to analyse the laminar flame front from a chemical point of view, we must remember that several chemical kinetic models and laminar flame velocity correlations have been carried out for its prediction and to roughly describe the behaviour of the mixture in the engine combustion chamber. As a first assumption to make some evaluations and to understand how the combustion event takes place, we assume that the system works under laminar flow conditions, in this way, there is the possibility to neglect the effect of corrosion due to turbulence, which complicates the analysis. Simplifying, it is possible to imagine the combustion chamber subdivided into two macro-zones, one where there are unburnt gases and the other where there are burnt gases, these are separated by a flame front which is supposed to be infinitesimal.

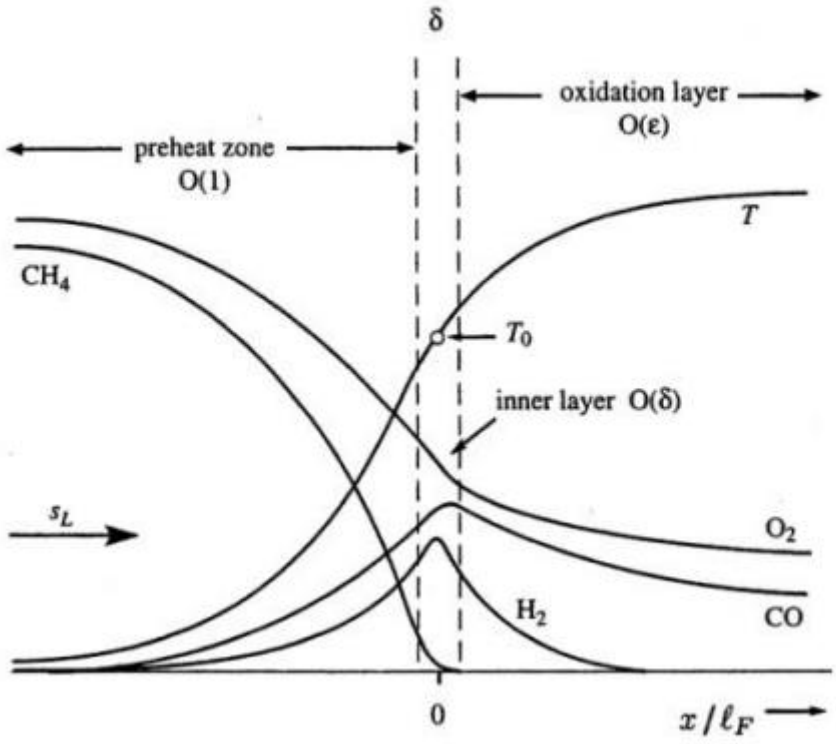

Figure 1.2.1: Trend of concentrations of different chemical species involved during methane combustion [16]

Figure 1.2.1 also shows the course of the various chemical species as they pass through the flame front. It is interesting to note that the laminar velocity is defined in a direction opposite to that of the propagation of the flame front (i.e. as if the front were stationary and the mixture was passing through it at a speed equal to the laminar velocity); this is because it is defined as the speed at which the reactants of the reactive mixture are consumed.

If we look at figure 4, with the flame front on a smaller scale, it is possible to distinguish three layers [16]:

- *Preheating zone*, where the available fuel is in the unburnt region, it reacts with oxygen to create O, OH and H radicals without releasing significant amounts of energy. We are in the initial phase promoted by the initiation activity of the plug. From the figure, it can be seen how the temperature gradient grows in an upward concave pattern due to the heat transferred from the inner layer.
- *Inner layer*, where the chain reactions mentioned above take place. In this phase the products of the reaction, i.e. the radicals, contribute to the reactants (fuel and radical) of another reaction, giving rise to a continuous transformation of products into reactants leading to a thermal explosion with a major release of energy.
- *Oxidation layer* characterised by the hot products at the equilibrium temperature of the burnt gases. In this zone, the temperature gradient has a concave tendency, tending downwards due to heat delivery.

All these reactions, which occur, are exothermic and the maximum for the rate of heat release is reached in the inner layer where the thermal explosion determines the combustion event.

We can use different types of approach to assess the flame speed, such as that in which the flame normally propagates and in relation to unburnt gases. In the latter case, an unstable spherical flame front propagates in the radial direction, and considering a one-dimensional flow in radial dimension, in scalar form, we can write the following formula:

$$\frac{dr_f}{dt} = V_u + S_{L,u} \tag{3}$$

Where:

- the first term represents the velocity of the flame front
- the second is the velocity of the unburned gas
- the third is the laminar combustion velocity measured at the unburned gas

Considering the principle of conservation of mass, applied between the burnt and unburnt zones, it is possible to calculate V_u:

$$V_u = \frac{\rho_u - \rho_b}{\rho_u} \frac{dr_f}{dt} \tag{4}$$

The term $\frac{\rho_u - \rho_b}{\rho_u}$ shows the expansion effect (unburnt gas density greater than the density of the burnt gas), a phenomenon which is present when the unburnt mass within the flame front becomes burnt. Consequently, V_b for assumption is equal to zero, because the burned mass inside the flame front is not moving, the laminar combustion velocity measured at the unburned gas is obtained:

$$S_{L,u} = \left(1 - \frac{\rho_u - \rho_b}{\rho_u}\right) \frac{dr_f}{dt} \tag{5}$$

For the simulation of combustion engine cycles, several models were evaluated [16]:

- *Zero-dimensional model*
- *Quasi-dimensional model*
- *Multi-dimensional model*
- *Multi-zone model*

The zero-dimensional model differs from the quasi-dimensional model due to the type of thermodynamic equilibrium, which gives us the possibility and ability to predict and visualise the three-dimensional phenomena occurring in the combustion chamber. The zero-dimensional model is based on two thermodynamic equations, which are the time-dependent mass conservation equation and the time-dependent energy conservation equation. Multi-zone models, in addition to the previous thermodynamic approach, include some geometric parameters, such as the interface radius, i.e. the radius of the flame that divides the burnt and unburnt gases, which is why the multi-zone model can also be called a two-zone model. In the latter, combustion is triggered by several identical and multiple zones of burnt gas which are generated at each specific crank angle and allow the flame to propagate and start. The figure 1.2.2 shows a multi-zone model for an SI engine, with a specific crank angle (θ), with an unburnt zone and six burnt zones, and the spark occurs at point one. Each zone of burnt and unburnt gas has a uniform temperature and composition, and a pressure that is shared evenly throughout the cylinder.

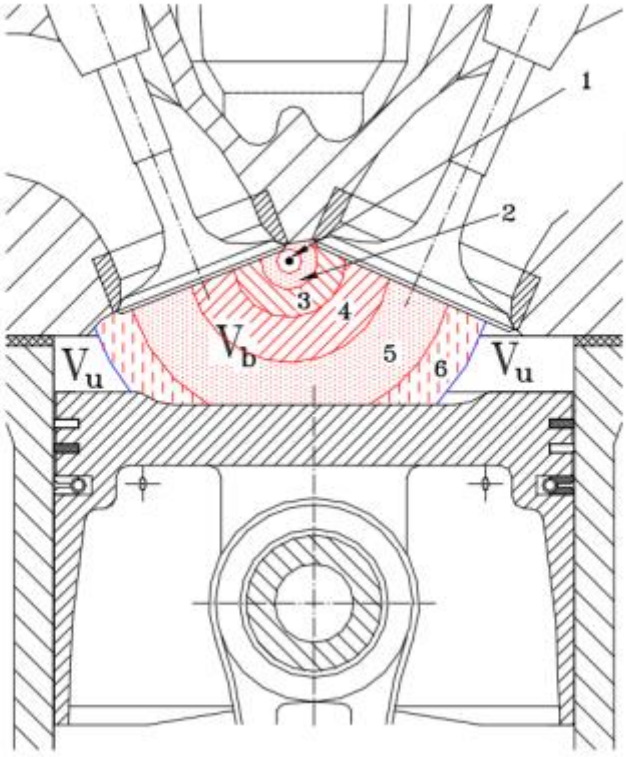

Figure 1.2.2: Representation of SI engine combustion event according to the Multi-zone model [16]

If we want to make a mass balance in multizone area, for the first thermodynamic law, we consider an ideal gas inside the combustion chamber, with a homogeneous mixture and composed of induced air, fuel and residual gases (the latter computed by means of a correlation in which the residual gases are in function engine speed and average pressure during the exhaust and intake pressure strokes) from the previous engine cycle:

$$m = m_a + m_f + m_r \tag{6}$$

For the conservation law, neglecting all possible leakages:

$$dm = d(m_a + m_f + m_r) = dm_u + dm_{b,n} = 0 \tag{7}$$

It is worth noting the presence of $dm_{b,n}$ in the formula: after the spark has been ignited, a first zone is burnt, this zone grows in volume and mass as a part of the unburnt gas is burning. At a given crank angle (θ), a new zone of burnt gas is created, while the previous zone created does not receive mass from the unburnt gas, but its volume continues to change due to its change in density, consequently we have that:

- dm_b is different from zero only in the last formed zone
- dV_b is always different from zero for each zone that is formed

If we now want to apply the energy conservation equation, it is necessary to write three different equations, one for the unburnt gas zone, n-equations one for each burnt gas zone and one for the last burnt zone:

a) $-q_u A_u \frac{d\theta}{\omega} + V_u d_p = (1 - x_b) m dh_u$ *for the unburned gas zone* (8)

b) $-q_{b,i} A_{b,i} \frac{d\theta}{\omega} + V_{b,i} d_p = x_{b,i} m dh_{b,i}$ *for the i-th burned gas zone* (9)

c) $-q_{b,n} A_{b,n} \frac{d\theta}{\omega} + V_{b,n} d_p = x_{b,n} m dh_{b,n} + m dh_{b,n} (h_{b,n} - h_u)$ *for the last b.g. zone* (10)

If we look at the three equations, we can see that in all of them appear the term "q" which is the heat flux transferred to the wall, the term "A" which is the surface through which the heat flux exchange takes place, the term "$d\theta\ /\omega$" is the time, the term "x" which is the mass fraction and "h" is the enthalpy. In all equations, we distinguish three terms:

- A first term is that of heat transfer
- A second term is the transferred work $pd\mathrm{p}$ (It is not pdV because on the right side of the equation there is the enthalpy term)
- An enthalpy change term relative to the specific zone to the right of the equal

In equation (10), there is an additional fourth term representing the presence of some unburned gases entrained in the burnt gas zone, which will change their chemical composition.

The multidimensional model is also governed by the Navier-Stokes equations, which also depend on space, as well as the conservation of mass and energy equation. However, in practical simulations with the simulation software, it uses analytical correlations as a function of the equivalence ratio, pressure, temperature and EGR value to obtain the laminar flame speed value. They are simply implemented in the simulation codes, considering that it is possible to obtain the laminar velocity value for any fuel type, once the mixture composition, pressure and temperature are known. Various forms of empirical and semi-empirical functional relationships have been proposed for the laminar burning rate, the most widely used being the power law formula:

$$S_L\,(\emptyset, T_u, p_u) = \ S_{L,0}\ \left(\frac{T_u}{T_0}\right)^{\alpha}\left(\frac{p_u}{p_0}\right)^{\beta} \tag{11}$$

where $S_{L,0}$ is the laminar velocity for a specific equivalence ratio (Ø), measured at (T_0, p_0) i.e. at ambient conditions, α and β are exponents that could be constant or dependent on the strength of the mixture. Gülder made the expression of $S_{L,0}$ explicit:

$$S_{L,0}\,(\emptyset) = Z\ W\ \emptyset^{\eta}\ \varepsilon^{-\xi(\emptyset-\delta)^2} \tag{12}$$

Where Z is equal to one for a single fuel constituting the mixture, and W, η and ξ are constants for a specific fuel. In the case of methane, the mixture is a mix of hydrocarbon molecules and their volume fraction varies, depending on the treatment applied during production, transport or the area where it was extracted [17]. Despite varying Z in the Gülder correlation, it has been shown that it is not sufficient to evaluate the effects of varying the composition of the natural gas on its laminar flame velocity [17], therefore evaluating the analysis of, Dirrenberger, who started from a mass balance, and obtained a valid relationship for a natural gas composed of methane, ethane and propane. The velocity estimated with this correlation has a good accuracy for lean and rich mixtures, but less acceptable near the stoichiometric condition. In the present thesis, we can refer to the laminar velocity measurement experimentally with Lowry, that use a constant pressure method (CPM), this is because our datasets were validated by comparing the simulated velocity values with the various experimental methods available for measuring laminar velocity, and the method with overlapping values was the CPM method. As for the Lowry estimation, it was found to be more accurate, even though it does not take into account the "radiation effect", neglecting the radiation at this stage of combustion can lead to large errors on the LFS. The hypothesis behind this choice is that the carbon dioxide and water produced by combustion are mainly responsible for this energy absorption and release mechanism; therefore, during the early stages of propagation there will be small quantities of these compounds in the chamber and therefore is negligible the energy absorption and the energy release. In other method, not considering the effect of radiation leads to an over-estimation of the LFS because the flame temperature would be higher than the actual temperature and therefore the flame front would propagate faster. The second aspect concerns the flotation effect, i.e. the creation of a vertical velocity component of the combustion gases due to the different density of the gases present in the chamber. This

effect is all the more significant the lower the velocity of propagation of the front, i.e. less than 15 cm/s. Lowry takes this other aspect into account compared to other methods, and this improves its accuracy. Furthermore, the heat losses, although not considered in Lowry, but being a CPM, for the first stages of combustion, are absolutely negligible (less than 1% of the total energy). The effect of flame front wrinkling can also significantly alter the final LFS value and this is taken into account in Lowry. Lastly, the pressure of the combustion gases, which is not taken into account in Lowry, because it is assumed that the front propagates in an unconfined environment, at least for the first moments of flame front propagation, and this would not influence much the error on the LFS, deriving from taking into account the pressure growth, which is less than 3% and therefore negligible.

1.3 Why choose natural gas?

Gaseous fuels are a good alternative to petrol because they have low reactivity. The main constituent of natural gas (CNG) is methane mixed with inert gases (N_2, CO_2) and ethane plus a small percentage of propane and high molecular weight hydrocarbons. CNG because it is stored and distributed in the form of compressed natural gas. Recently, technologies for storing natural gas in liquid form (LNG) are also becoming more widespread. LNG is a viable alternative for heavy-duty traction vehicles that need to meet requirements on vehicle range and are characterised by high fuel consumption. There are numerous advantages of natural gas over petrol:

- Greater availability
- Reduction of CO_2 because for the same amount of energy obtained from the combustion process, methane has a lower carbon content than petrol, and greater efficiency in the extraction-refining-distribution chain, which translates into benefits in terms of a well-to-wheel analysis.
- The compact molecule makes it highly resistant to detonation. This is why it is difficult to calculate the octane rating, as it would be on a scale of around 130 octane. A wide range and variety of octane numbers depends on the composition of the fuel, which allows a higher compression ratio to be used than for petrol with benefits in terms of efficiency and CO_2 emissions. However, the problem with these engines is the inefficient storage of the fuel, which is in gaseous form. This is because gaseous fuels have a low energy density and therefore need a greater volume to achieve the same range as a vehicle powered by liquid fuels. Another problem is that distribution is less widespread than with petrol. Because of these problems, bi-fuel applications have become widespread, i.e. engines that run on both petrol and natural gas, but the design of a bi-fuel engine requires a lower compression ratio for petrol operation to avoid detonations, and so we will not exploit the full potential of methane. In figure 6 we can see what happened with a bi-fuel engine.
- Since the fuel is gaseous and not liquid, there is no need to enrich during transients, thus making the engine run with a stoichiometric ratio equal to one. This has benefits in terms of CO_2 emissions, lower fuel consumption, and lower carbon monoxide and hydrocarbon emissions.
- Unburned hydrocarbons are less aggressive and have less impact on the environment. All hydrocarbons are reactive in the atmosphere, especially with nitrogen oxides, resulting in photochemical smog. However, methane hydrocarbons are less reactive due to the rigid and compact form of the molecule.
- The natural gas is lighter than air, so there is no particular risk in the event of a gas leak, as is the case with LPG (liquefied petroleum gas), which is heavier than air.
- Natural gas has a very low flammability range and there is little chance that a flammable mixture will be created by methane leaks.

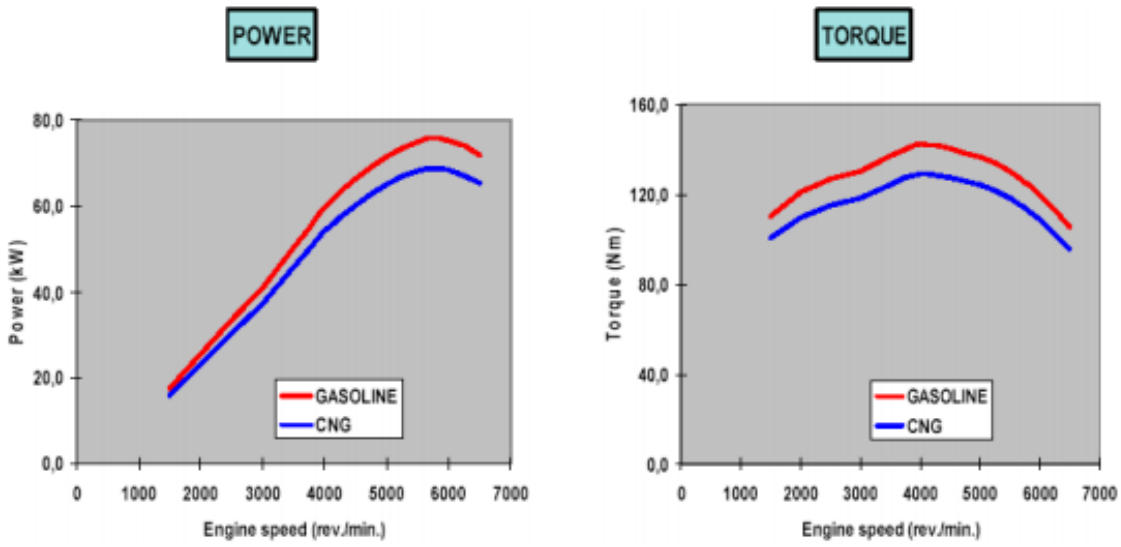

Figure 1.3.1: Same engine running on petrol and natural gas, the engine cannot fully exploit the natural gas [14]

In addition, it does not have particularly high LFS values, which is one of the reasons why it was decided to add hydrogen, in order to solve this problem, allowing a shorter combustion phase duration and thus increasing engine efficiency. The direct drawback of adding hydrogen is the achievement of high temperatures in the end of the combustion chamber, and thus emissions, the production of which depends mainly on a thermal mechanism that is strongly influenced by the temperatures reached in the combustion chamber. In addition, another drawback could be caused by the fact that the engine is in a gaseous state. In fact, in traditional spark-ignition engines, using a rich fuel reduces the temperature in the chamber thanks to the heat subtracted for the vaporisation of the fuel. In the case of methane, this effect could not be relied on to keep chamber temperatures down.

1.4 Why use methane-hydrogen mixtures?

An alternative to conventional fuels is mixtures of hydrogen and methane. This combines the advantages of methane with those of hydrogen, combining a higher combustion speed with a high resistance to detonation. In fact, methane has a low front propagation speed for mixtures even slightly far from stoichiometric. One could think of intensifying the turbulence in the chamber in order to compensate for the low values of front propagation speed, exploiting a duct geometry that improves turbulence, but the filling coefficient will be affected, since the entire cross-section of the duct would not be used. The solution is to use hydrogen as an additive, which also reduces the cyclical variability of the engine, operation at partial loads and also reduces incomplete combustion. The main differences between the two fuels are

- the laminar flame speed, which is about ten times higher for hydrogen, and the flammability limit, which is much higher for hydrogen.
- A particularly interesting parameter is the energy density, which is clearly in favour of methane, meaning that, due to the low density of hydrogen, a large quantity of fuel needs to be stored on board the vehicle.
- hydrogen has a very high stoichiometric ratio, which means that large quantities of air must be introduced into the combustion chamber to ensure that the engine operates at the required stoichiometric ratio. Therefore, when creating the mixture of methane and hydrogen, a compromise has to be made between improving combustion due to the addition of hydrogen and the simultaneous decrease in energy density.

The addition of hydrogen allows the combustion phase to be completed in ever shorter intervals of time, but with the consequent achievement of ever higher temperatures (combustion tends towards the higher temperature). The combustion tends to the ideal isochoric one foreseen by the Otto cycle), with a consequent increase in NOx emissions.

2 Introduction on Machine Learning and Artificial Neural Networks

One of the first definitions of machine learning is given to us by Arthur Samuel who defines it as " the field of study that gives computers the ability to learn without being explicitly programmed."

Nowadays, machine learning, and more generally neural networks, are widely used in the fields of robotics, medical and networking, while it remains a topical subject in the automotive field, where, only a few years ago, they started to be investigated to bring improvements in terms of safety, control and emissions of cars. Machine Learning (ML) is the science (and art) of programming computers in such a way that they can learn from data [2]. In the present work of the thesis a code is used, written in Python environment, which exploits an algorithm of Machine Learning, specifically called Linear Regression. We will see later in detail how this algorithm is made up and how it works.

Machine learning uses various statistical methods to learn directly from data. The first real step, where the available data is examined to create a model, is 'training'. From the analysis and training, an algorithm is obtained to represent our dataset. Once the algorithm is found, it is used to make predictions and estimates. The loop is represented in the illustration below:

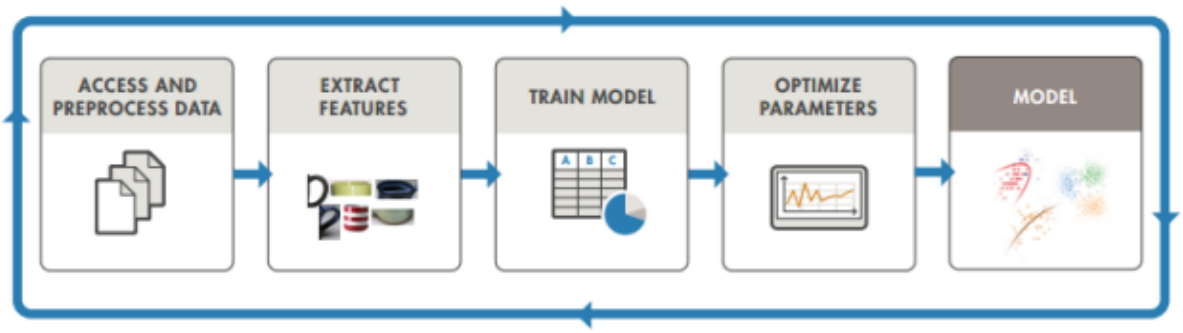

Figure 2.1: Machine learning loop [1]

The algorithm that is reflected in the training must be guided to look for hidden correlations, thanks to features that are provided by the input data. The characteristics on which our algorithm must train, and those which it must predict, are then provided. No less important is the next step, the parameters and hyperparameters optimization to extract a better result from our algorithm. The goal is to have a final model that satisfies our search parameters.

2.1 Classification of machine learning algorithms

There are different types of Machine Learning, they can be divided into three macro categories:

- the category in which they can or cannot do training with human supervision, and in this domain we find supervised, unsupervised, semi-supervised and Reinforcement Learning;
- the algorithm due to training may or may not learn exponentially during application;
- Whether they work by simply comparing new data to already known data, or
- they detect patterns using training data and then build a predictive model, a scientifically similar approach.[2]

These criteria can also be combined. For this thesis work we focus on the first macro-category, the reason for this choice will be explained later.

Let's briefly look at the Machine Learning systems that can be classified according to the type and amount of supervision they can obtain during training.

2.1.1 Supervised Learning

In supervised learning data are provided in 'input-output' form, are used to extract a 'general rule' associating them, the data that we feed includes the labels which is our goal. This method is based on the experience gained in the training phase. Then the algorithm is able to create an output for an input it has never seen before without the help of a human being. We have two areas where supervised learning can be applied:

- Classification: the learning system must create a model that filters the inputs, the algorithm is first trained on several unknown data, each belonging to a different class, and then learns to classify the new ones;
- Regression: from multiple inputs, the learning system must predict a single output. This is a different approach where the algorithm itself must predict a certain numerical target value, given a certain set of features, called predictors. To train such a system it is necessary to provide, in general, a lot of data containing both predictors and target values, called labels, which can be, therefore, considered as input and output of the system respectively. Thus, solving a regression problem corresponds to learning an approximate function of the given input-output pairs.

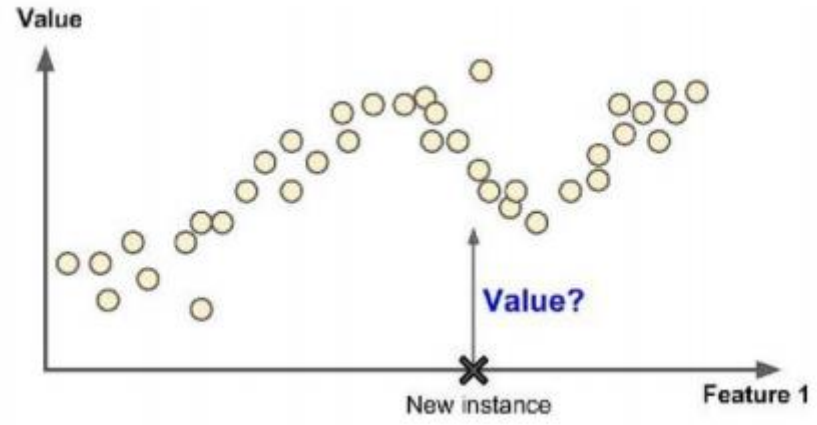

Figure 2.1.1.1: Regression of known data and prediction of a new instance [2]

The most important and well-known supervised learning algorithms, of which we will only discuss the one of interest for the present work, are:

- K-Nearest Neighbors

- Linear Regression

- Logistic Regression

- Support Vector Machines

- Decision Trees and Random Forests

- Neural networks

Among these, as previously mentioned, we will analyse only the case of Linear Regression used for this work.

2.1.2 Unsupervised Learning

In unsupervised learning algorithms, the training set is not labelled, i.e. it does not contain the output values by which to train the model and, therefore, the pattern classes used for training are not known. The algorithm must find a structure in the input data, sort and learn. It is used to identify a 'cluster' or grouping. The most important algorithms of this type include:

- Clustering: identifies groups (clusters) of patterns with similar characteristics in which the classes of the problem are not known and the patterns unlabelled (no output). Often the number of clusters is not known a priori, but those identified in the learning can then be used as classes. The unsupervised nature of the problem makes them more complicated to classify.

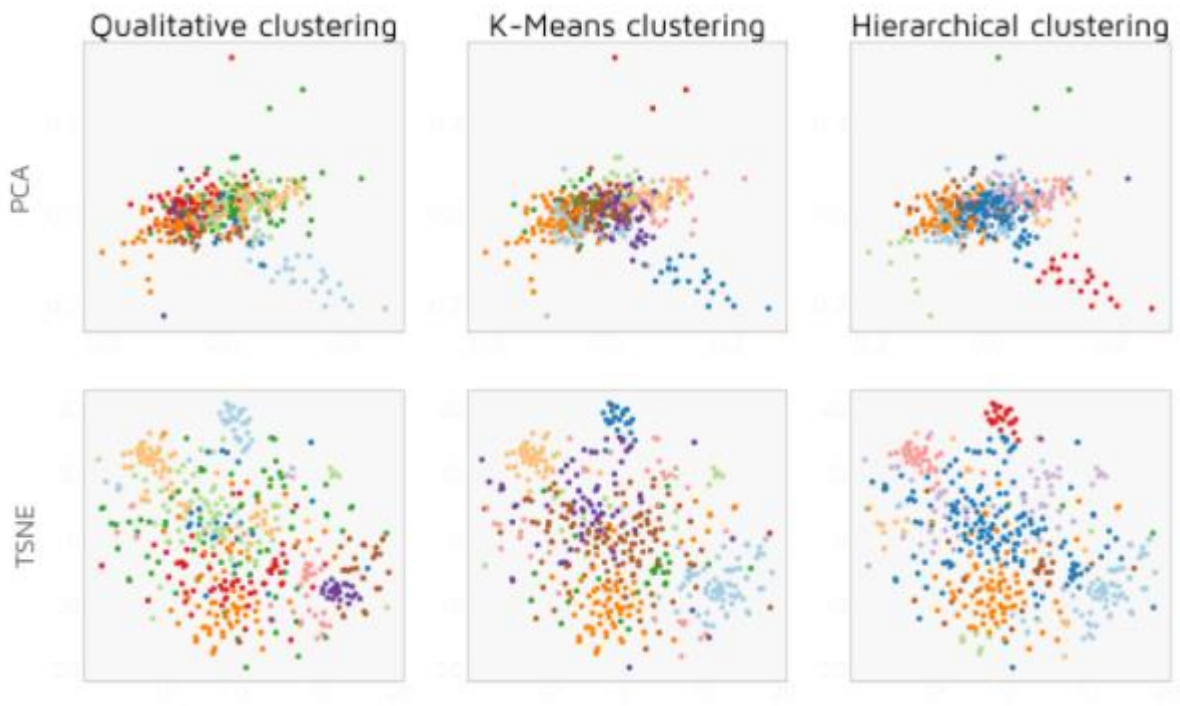

Figure 2.1.2.1: Type of clustering. PCA stands for Principal Component Analysis. TSNE stands for t-distributed Stochastic Neighbor Embedding [3]

- Dimensionality reduction: reduces the number of dimensions of the input patterns, without losing too much information. The operation involves a loss of information, but the aim is to keep the information that is important for the case, and therefore dependent on the application. It is therefore very useful to make very high dimensionality problems tractable, discarding redundant and/or unstable information, thus allowing the code to compute faster due to the smaller disk space disk space occupied by the data and sometimes higher performance.

2.1.3 Semisupervised Learning

Semi-supervised learning algorithms work with only partially labelled training sets and the distribution of patterns without output can help to optimise the classification rule. Many algorithms of this type are combinations of supervised and unsupervised. An example of the use of semisupervised learning is in the

photo-hosting services algorithm, where by uploading several photos of groups of people, we obtain clustering (unsupervised learning) of people A who appear in photos 1,3 and 5 for example, while people B are identified in photos 2, 4 and 6. [2]

2.1.4 Reinforcement Learning

This is a machine learning technique that aims to create autonomous agents capable of choosing actions to achieve certain objectives through interaction with their environment. A Reinforcement Learning algorithm targets the learning of optimal behaviour from past experience. The acquisition of an agent, which observes its environment and performs actions to modify it. Actions to modify it, causing transitions from one state to another and receiving “rewards”. Rewards, which may also be penalties in the sense of negative rewards. The goal is to learn the optimal action in each state in order to maximise the sum of the rewards obtained in the long run. It is important to underline an aspect that concerns the ML in general: since the main objective is to select a learning algorithm and train it on a certain set of data, the selection is extremely important since we may have a "bad algorithm" and "bad data", in the sense that they may not be compatible, or we have only a small number of data available for training, or even the chosen algorithm is not compatible with the application itself.

2.2 Artificial Neural Networks and Deep Neural Networks

just as man took inspiration from birds for the field of aviation, so too in the field of artificial intelligence he took inspiration from the architecture of human’s brain. This was the key that led to the development of the artificial neural network (ANN). ANN are a supervised learning system made up of a large number of simple elements, called neurons or perceptron. Any neuron can make a number of simple decisions, and feeding those decisions to other neurons, organised in interlinked layers [4]. In other word ANN is a machine learning approach, seen as a “mathematical model”, formed by artificial neurons used to solve problems. A "simple neural network" is formed by:

- *Layer of input* neurons (or nodes), which receives signals from outside
- *Hidden layers* that receive signals from the input layer or from other intermediate layers
- *Output layer*

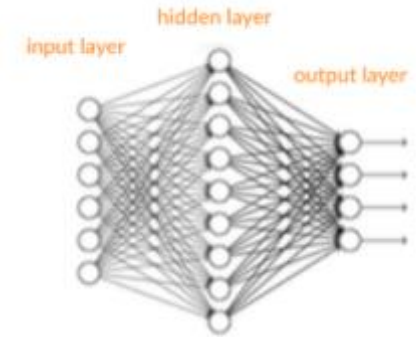

Figure 2.2.1: Network’s architecture

And we can say that ANN is the real beating heart of Deep Learning. In this chapter we introduce the architectures of the ANN up to the Deep Learning.

2.2.1 From Perceptron to Deep Neural Networks

Cornell Frank Rosenblatt invented the first architecture of a perceptron in 1957, the first binary classification algorithm [2]. It helps to divide a set of input signals into two parts: "0" and "1", i.e. "yes" and "no". But unlike many other classification algorithms, the perceptron is designed around the essential unit of the human brain, the neuron, and has an extraordinary ability to learn and solve complex problems. A perceptron is a very simple learning machine. It can take a small number of inputs, each of which has a weight to indicate how much it matters, and generate an output decision of '0' or '1'. However, when it is combined with many other perceptron, it forms an ANN.

To solve more complicated problem we need a perceptron network. Referring to figure 2.2.1.1, it shows an ANN called multi-layer perceptron (MLP), with exactly three layers. Each perceptron in the input layer, sends outputs to all the perceptrons in the hidden layer, and all perceptrons in the second layer send outputs to the the output layer. From each perceptron there are several input signals, each signal goes to each perceptron of the hidden layer, but these signals use different weights, so we will have different outputs. In figure 2.2.1.1, each line going from a perceptron from one layer to the next layer carries a different output. In this way it is possible to obtain a complex system, i.e. the neural network, consisting of several perceptrons and several layers. An MLP with three layers, such as the diagram above, is called a non-deep neural network. An MLP with more than four layers is called a Deep Neural Network (DNN). Unlike the classic perceptron, the neural networks obtained by MLP, can use functions and weights able to have outputs also different from 0 and 1, typical characteristic of "simple" problems of classification, and then obtain values in the real field.

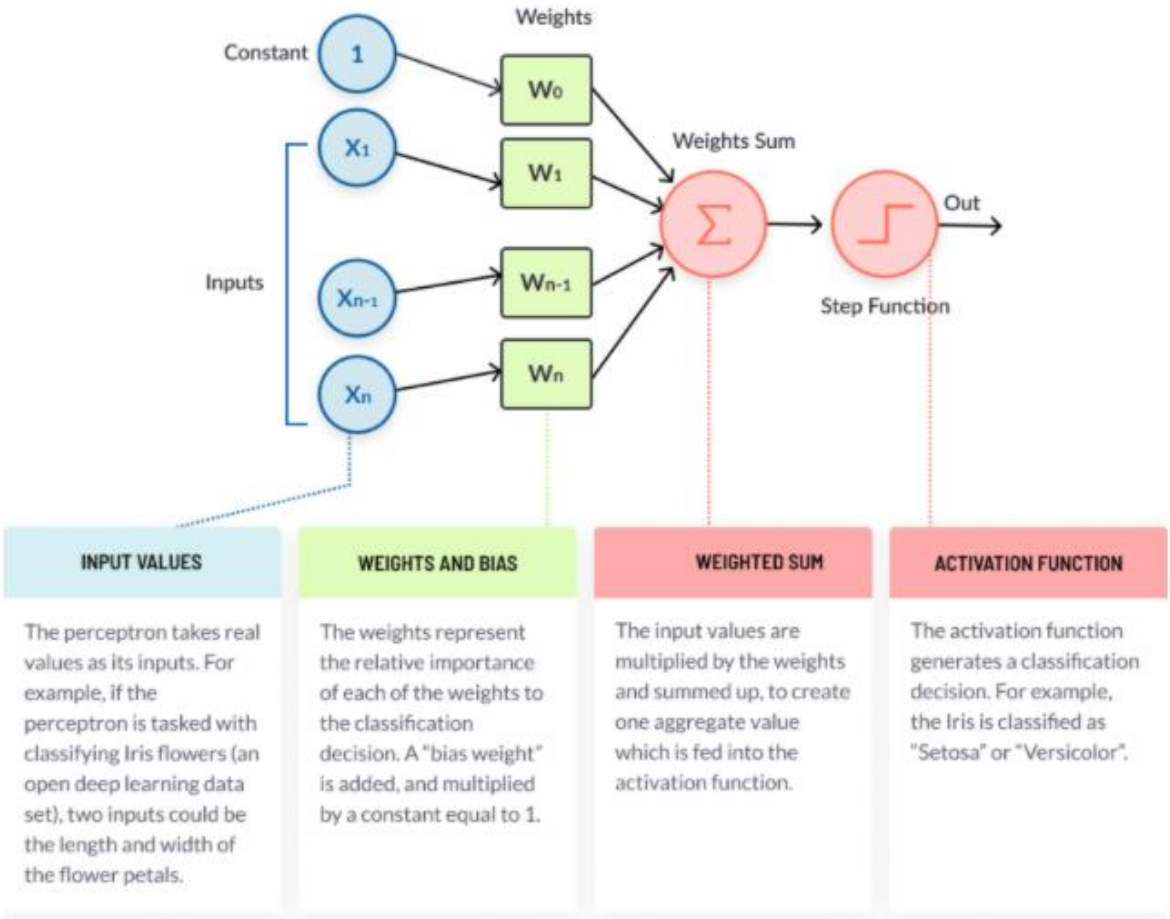

Figure 2.2.1.1: Perceptron's structure [4]

2.3 Data available for our neural network

The starting data, which will be used to 'feed' our first layer of the neural network, were obtained using LOGE research, a LOGE tool for studying reactive flows using complex chemical kinetics models. It allows combustions to be simulated using detailed chemical mechanisms for a wide variety of engineering applications; it also allows different types of reactors to be simulated, as well as combustion in the combustion chamber. In fact, the laminar burning speed can be either measured (using different experimental procedures) or calculated by simulating the chemical kinetics of the reactions involved in the entire combustion process. In this thesis work, an attempt was made to predict the laminar burning speed by means of a deep learning algorithm that identifies the connections between the initial simulation input data and the simulation result (i.e. the speed). Two large libraries of LS values, measured in [cm/s], obtained under different initial conditions and by two different chemical mechanisms, were analysed:

- ARAMCO 2.0
- GRI-mech 3.0

The libraries were obtained by varying various physical parameters, in particular:

- Dosage [-]
- EGR [%]
- Temperature [K]
- Pressure [bar]

2.3.1 Aramco 2.0 library

The first step involves the creation and the analysis of a library of LS at the variation of EGR, dosage, pressure and temperature. In Table 2.3.1.1, the ranges of the quantities under consideration are summarised.

Parameters	Range	Discretization step	Number of points
Initial pressure [bar]	1-200	From 1 to 20 bar ⟶ step 1 bar From 20 to 100 bar ⟶ step 10 bar From 100 to 200 bar ⟶ step 20 bar	33
Temperature [K]	300-1100	100 K	9
Dosage [-]	1	-	1
EGR [%]	0-50	10%	6

Table 2.3.1.1: Range of Aramco 2.0 parameters

So in total 1782 different cases were simulated. If we vary the dosage in a range from 0.75 to 1.25, in steps of 0.05, i.e. 10 steps of 0.05[-] the total number of simulated points is 17820.

The available data had a dosage value of 1 for all simulations. It was decided in this case to create a single dataset, eliminating the "phi" column because it did not give a real contribution to the training phase of our network. Furthermore, observing the data, included in our dataset, for high values of pressure and temperature, presented "NaN" velocity values (not numbers), it was decided in this case to delete the whole respective row. The decision was made after testing the neural network, and observing a not optimal training of this, specifically it was noticed that the network learned in a wrong way, showing "prediction" NaN values in the initial training phase. For this reason, upstream of our neural network, an algorithm of classification was elaborated, that is, a network "pass/fail", which discriminates the NaN values. As a goal it must give as a result 1 when it finds a value of NaN speed, and 0 when it finds a real value of speed. This point will be taken up in the chapter dedicated to the description of the neural network.

A single dataset was created, resulting in a file of 1626 rows and 4 columns, as we can see from table 2.3.1.2, we have a recap of min and max value, standard deviation and mean value:

	EGR [%]	p [bar]	T [K]	Speed [cm/s]
Rows	1626	1626	1626	1626
Columns	4	4	4	4
Mean	23.77	46.24	725.83	50.37
Std	16.72	5.57	247.89	66.85
Min	0	1	300	0.088
Max	50	200	1100	480.90

Table 2.3.1.2: Cleaned available Aramco 2.0 data

The data analysed shows that the average speed of our file is around 50.37 cm/s, a value that we must keep in mind, when we go to evaluate the performances of our neural network. This is because we expect that our network, which learns and trains from our input file, will be more accurate at low to medium speed values.

A further analysis to confirm the distribution of our data can be observed from the figure 2.3.1.1, where we find a summary of the trends and the distributions of our data contained in our dataset.

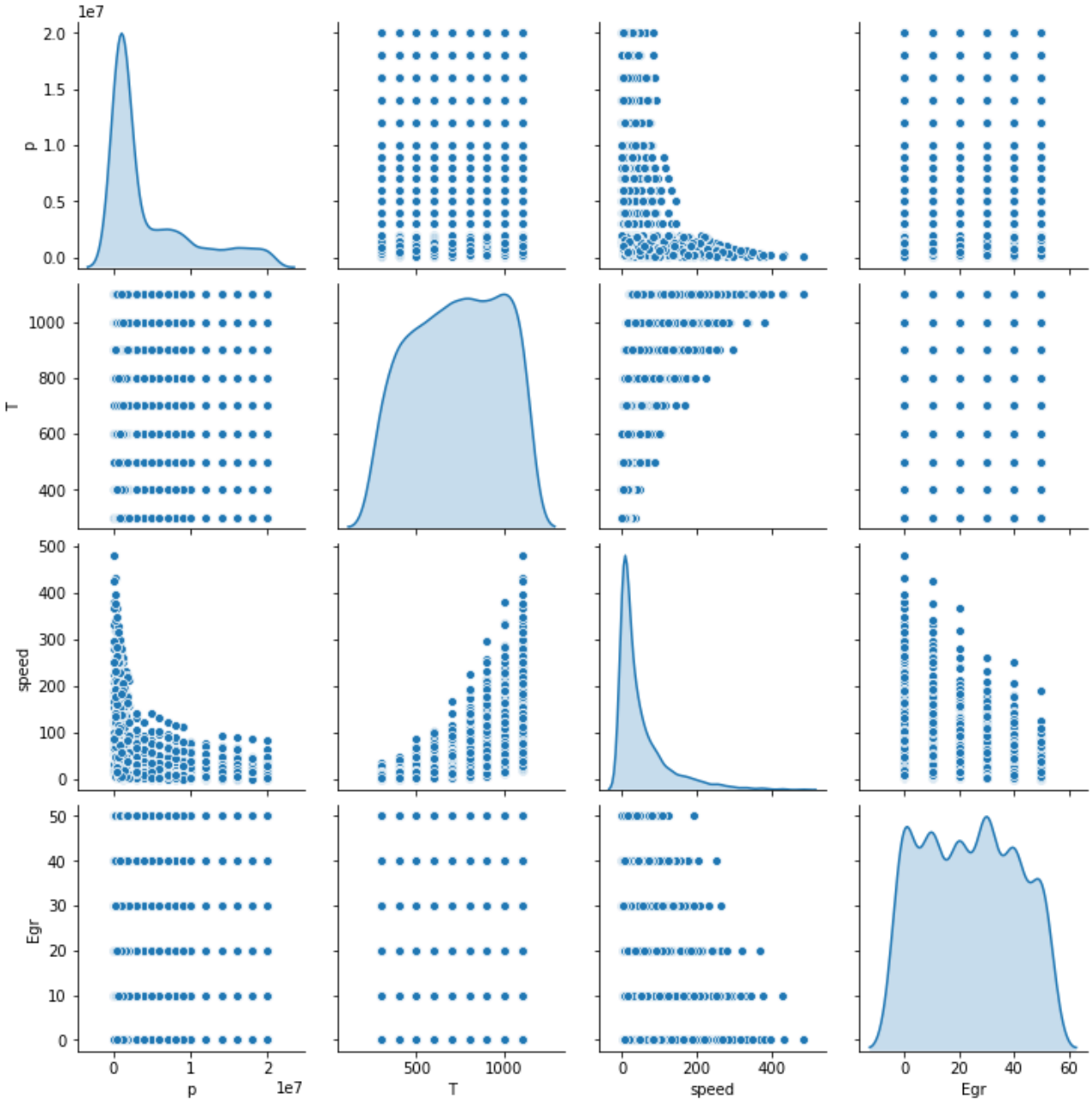

Figure 2.3.1.1: Dataset Aramco 2.0 data distributions

2.3.2 GRI-mech 3.0 library

Also in this case, the first step involves the creation and the analysis of a library of LS at the variation of EGR, dosage, pressure and temperature. In Table 2.3.2.1, the ranges of the quantities under consideration are summarised.

Parameters	Range	Discretization step	Number of points
Initial pressure [bar]	1-200	From 1 to 10 bar ⟶ step 5 bar From 10 to 20 bar ⟶ step 10 bar From 20 to 40 bar ⟶ step 20 bar From 40 to 200 bar ⟶ step 40 bar	9
Temperature [K]	300-1600	From 300 to 1200 K ⟶ step 100k One point at 1600 K	11
Dosage [-]	0-5	From 0 to 2 ⟶ step 0.1 From 2 to 3 ⟶ step 0.5 One point at 5	23
EGR [-]	0-0.6	0.1	7

Table 2.3.2.1: Range of Gri-mech 3.0 parameters

So in total 15939 different cases were simulated.

Again, a single txt file was created from the results of the simulations carried out. It was cleaned from the 1.0000000E-03 speed values in the file, as they were NaN values. These non-numbers, as can be imagined, came from zero dosing values, as was obvious to expect, and from high stoichiometric ratio values, i.e. simulations with phi equal to 5, and as in the previous case we have non-numbers when you have very high pressures and temperatures. Once we have cleaned up our file, this resulting with 11602 rows and 5 columns. We move on to the analysis of it, through mean value, standard deviation, min and max value ,as shown in table 2.3.2.2:

	EGR [-]	p [bar]	T [K]	Phi [-]	Speed [m/s]
Rows	11602	11602	11602	11602	11602
Columns	5	5	5	5	5
Mean	0.25	70.41	846.43	1.29	0.69
Std	0.18	69.53	364.90	0.7	1.56
Min	0	1	300	0.2	0.001
Max	0.6	200	1600	3	26.46

Table 2.3.2.2: Cleaned available Gri-mech 3.0 data

In contrast to table 2.3.1.2, we can see that the speed is not expressed in cm/s but in m/s. Also we can notice that the EGR values are not expressed in percentage. What is very important is that we have a dataset with dosage values not equal to one, but a total of 23 intermediate points between the min and max values. We stress these differences because it will be very important to see how our first attempt neural network will behave with a dataset that is substantially very different from the Aramco 2.0 dataset. A further analysis to confirm the distribution of our data can be observed from the figure 2.3.1.1, where we find a summary of the trends and the distributions of our data contained in our dataset. From the figure, we can see that the simulations were launched by trying to take the pressure, temperature and dosage values to extremes. It is worth noting that there is a grouping of points which present a high speed value, approximately 25 m/s, with:

- low pressure, close to 1 bar;
- temperature of 1600 K;
- EGR set at 0.4;
- dosage into the range 0.2-0.3.

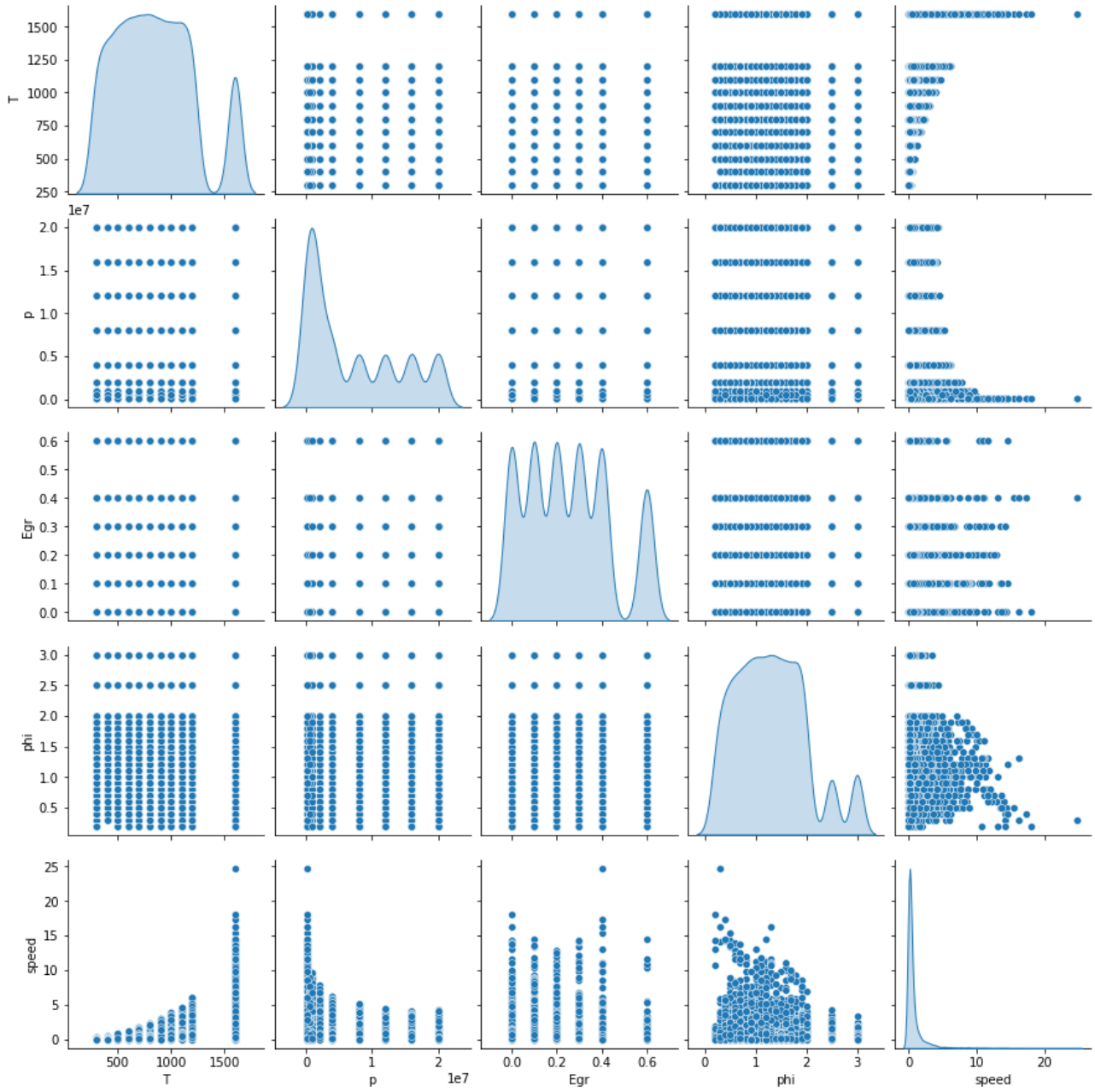

Figure 2.3.2.1: GRI-mech 3.0 dataset data distributions

This result is to be investigated with the formulas of chemical kinetics with a comparison through the Aramco simulation using the same step size. The above points were investigated, with the first point being the highlighted one. Unfortunately, the Aramco 2.0 software did not report any real results, nor even a NaN value, it could not interpolate this combination of points, so it is impossible to make a direct comparison between the two mechanisms and the kinematics. It should be remembered that the Aramco dataset previously illustrated was validated using Lowry's kinematic method, taking representative points throughout the dataset and seeing how it approached the values of the chemical kinetics. It was therefore decided to continue with this dataset, because if we were to make a comparison we would need to have results from both modes of simulation, with the same step size. This is time-consuming as Aramco is more accurate but requires much more computational time. Furthermore, the aim of our thesis work is to see if it is possible to find relationships between the various input data, even non-linear ones, using machine learning techniques, and this dataset represents a real challenge for our algorithm.

2.3.3 Comparison of Aramco 2.0 and GRI-mech 3.0

From the libraries, it was possible to compare the two mechanisms and to specify the differences between the mechanisms under analysis, which are summarised in the following table 2.3.3.1:

Kinetic Mechanism	**GRI-Mech 3.0**	**Aramco 2.0**
Composition	Includes CH4/C2H6/C3H8 Mixtures	Includes CH4/C2H6/C3H8/C2H4/C2H2/ C2H5OH/C3H6/CH3OH Mixtures
Experimental techniques	Rapid compression machine and shock tube	Shock tubes, rapid compression machines, flames, jet-stirred and plug-flow reactors
Pressure range of validation	13-21 bar	1.2-260 bar
Temperature range of validation	850-925 K	1040-2584 K
Chemical reaction and species	325 reactions and 53 species	2716 reaction and 502 species

Table 2.3.3.1: Differences between Aramco 2.0 and GRI-Mech 3.0 mechanisms [5]

The two mechanisms are derived from the experimental data, by setting some coefficients of the equations solved by the chemical model, in order to reach the same values of LFS obtained experimentally. The main difference between the two reaction mechanisms is the number of chemical species and reactions with which the combustion phenomenon is simulated, in particular Aramco 2.0 uses about ten times the number of species and reactions used by GRImech 3.0. The reason for that approach is found in sensitivity consideration, i.e. not all the reaction have an equal weight. As consequence the simulation times increase exponentially with the number of reactions and species contained in the mechanism, and the choice number of reaction are a consequence of compromise between time and accuracy. We aspect a great precision from Aramco 2.0 and a faster simulation from GRI-mech 3.0.

2.4 Data processing and normalization techniques

It is good practice to normalise features using different ranges and scales, in our case we have different ranges of pressure, temperature, dosing and EGR, all ranging from low to high values, as seen in the previous chapter. Our model receiving, the input data, although it can converge without normalising the features, but this makes training more difficult, and makes the resulting model dependent on the choice of units used in the input. Below we will look at some of the data processing and normalisation techniques. You want to get all the data on the same scale, this is because if the scales for different features are vastly different, this can have a knock-on effect on your ability to learn (depending on the methods you are using to do this). Data normalisation must ensure that standardised function values implicitly weight all features in their representation.

2.4.1 Normalization and standardization

Normalization and standardisation are two processes used in the data pre-processing phase in which the data is prepared for further processing by our neural network. The two methods seek to scale the data set. Normalisation is a re-scaling technique in which we shift and re-scale values so that they end up oscillating between 0 and 1. It is also commonly known as Min-Max scaling [6]. From the figure 2.4.1.1 below and formula (13), we can understand how min-max scaling works.

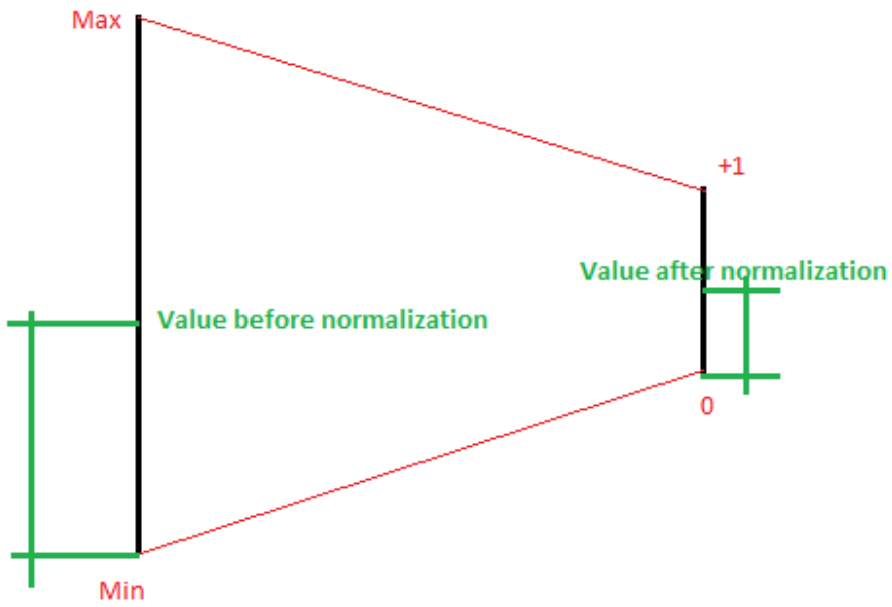

Figure 2.4.1.1: Normalization effect [7]

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (13)$$

Where X_{max} and X_{min} are the maximum and minimum values.

Standardisation is another scalability technique in which values are centred around the mean with a standard deviation equal to one. This results in that the mean of the attribute becomes zero and the resulted distribution has a unit standard deviation [6]. The standardization formula (14) is described as:

$$X' = \frac{X - \mu}{\sigma} \quad (14)$$

Where μ is the mean values and σ is the standard deviation. In this case we have not a restricted range as in case (13). Now it's important to understand if for our distribution data which is better. Normally the general rule say that Normalization is most used when we know that the data not follow a Gaussian distribution, but show a random distribution data. At the contrary standardization is used when we have data that are described which a Gaussian curve. However, the choice of whether to use normalisation or standardisation depends on the problem and on the machine learning algorithm used. There is no fixed rule. As will be made clear later, our datasets will be divided into training, testing and validation, these serve respectively to train our network, to test if it fits well and to validate the hyperparameters. Therefore, a good practice is to fit the scaler to the training data and then use it to transform the test data, avoiding any data loss during the model testing process. Moreover, normalisation or standardization of target values is generally not required. In this case we decided to use the (14) formula applied to our dataset with some differences:

normed_trainData = (trainData - mean(trainData)) / std(trainData) (15)

normed_testData = (testData - mean(trainData)) / std(trainData) (16)

We chose to normalise our data using only the training data. This choice depends on the size of the datasets and whether both the training and the test are equally representative of the domain you are trying to model. If you have thousands of data points and the test set is completely representative of the training set (although this is difficult to prove), then you will be able to normalise the training set with the training data and the test set with its respective data. If you use a small but representative set of test data, then it is best to normalise using only the training parameters, as sampling errors can negatively affect predictions. We consider in our choice the scaling process as part of the model generated from the training data, so the test data is testing both the generality of the model combined with the pre-processing. Very often people scale all the data and then split it into training/test datasets, but in this case, the test will then validate the model on its own, which can be useful if your goal is not to produce a predictive algorithm but to understand the structure of the data (i.e. the important variables) [8]. Our conservative choice was made by looking at the size of the data sets and whether both the train and the test are equally representative of the domain you are trying to model. Having two datasets with different data and different sizes, this conservative way was chosen. The available data has been standardised to prevent the network only learning from values that are too large or too low.

2.5 Description of our deep learning algorithm

As mentioned previously, for the realisation of our neural network we will use linear regression, which is used in science but also in finance, for predictive problems, and can also be used for deep learning problems. What is the connection between neural network and linear regression? Let us look at a simple regression equation:

$$\gamma = \beta_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_k X_k + \varepsilon \quad (17)$$

Where:

- γ is the dependent variable (the value to be predicted)
- X_k is the independent variable (input values)
- β_k are the weights or correlations to be found by the network
- ε is the error, i.e. the distance between the predicted value and variable γ

Neural networks are able to model complex problems, using a learning process that simulates the human brain. The neural network generates a model that approximates any regression function. In addition, many of the regression models do not fit the data perfectly, and neural networks must therefore generate a more complex model that will provide greater accuracy. Therefore, it was decided to use linear regression to build our deep learning algorithm.

2.5.1 Training validation and test

The best way to know how a model will adapt to new data is to test it on the instances themselves. To do this the dataset supplied to the algorithm is divided into two parts: Training and Testing, the first being split into training set and validation set:

- Training test: data set used for learning, to fit the parameters of the model (linear regression), we would use the training set to find the "optimal" weights β.
- Validation test: set of patterns on which the system tares the hyperparameters that will be used to optimise our algorithm. Used to see if the chosen model fits our case. In our case, we would use the validation set to find the "optimal" number of hidden units or determine a stopping point for learning the algorithm.
- Testing: set of patterns on which the algorithm evaluates the final performance. It is only used once a model is completely trained.

Why separate test set and validation? A general rule say that the estimated error rate, of the final model, on the validation data will be biased (lower than the real error rate) because the validation set is used to select and evaluate the model. After evaluating the final model on the test set, the further model should not be tuned!

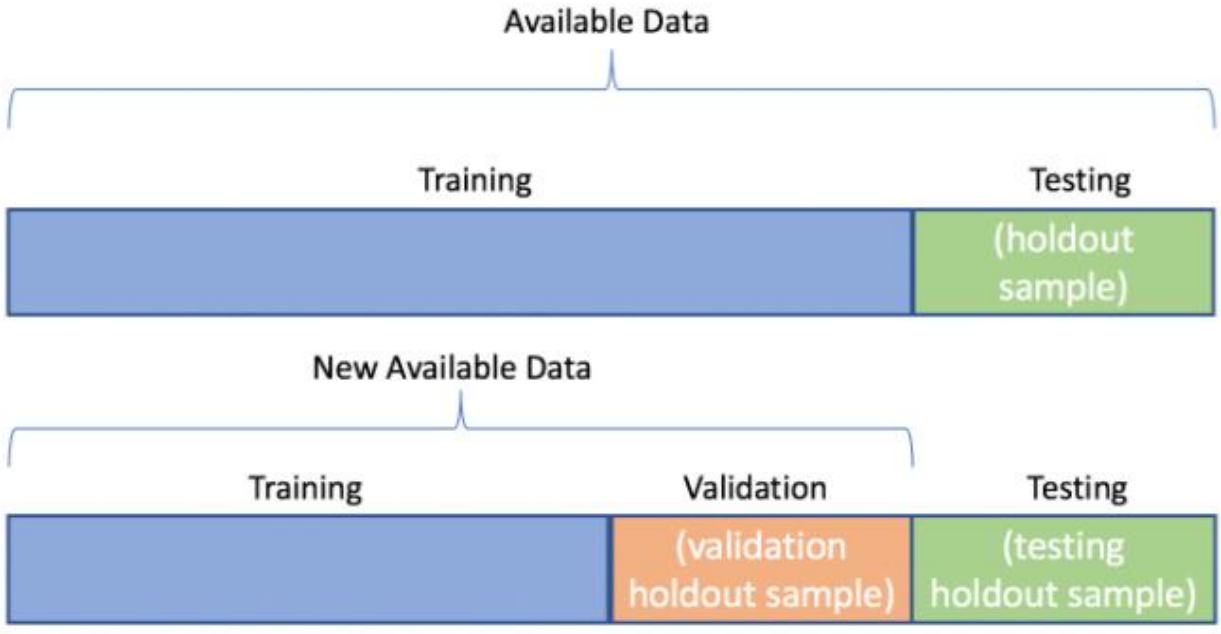

Figure 2.5.1.1: Dataset subdivision obtained by the algorithm [9]

From the previous division, evaluating the model on the test set, we obtain an error rate on new instances, which is called the generalisation error. This value tells us how well our model will perform on instances it

has never seen before. We will use 80% of the data for training and 20% for testing. The validation set will be proportionally the same as the test set but will be included in the training data. The need to have a test set, unrelated to our network that is learning from the training data, and a validation set results from the fact that if we use the test set to find the ideal hyperparameters for our case, our network will underestimate the generalisation error, because it optimises and then evaluates itself on the test set. This means that the model is unlikely to perform as well on new data [2]. In general, the behaviour of a Machine Learning algorithm is governed by a set of parameters that characterise it. The learning consists in determining the optimal value of these same parameters. Therefore, given a training set Train and a set of parameters, the objective function. The majority of the algorithms require to define, before the real learning, the value of the so-called hyper-parameters and, once appropriately chosen, the learning is carried out for each one, going to take those that have provided the best performances. Examples of hyper-parameters can be the number of neurons in a neural network, the type of loss-function, the number of levels, optimisation functions and so on. Let us then give a definition of what is considered to be the performance of the algorithm and define the parameters that characterise it.

2.5.2 How to evaluate our deep neural network

All machine learning algorithms aim to maximise or minimise a function, which we call the "objective function". The latter if they are minimised are called "loss functions", they are a measure of how well a model does. Another method used to get the minimum point of the function, is "gradient descent". The loss function is like an undulating mountain and gradient descent tries to find the lowest point. There is no single loss function that works for all types of data, it depends on a number of factors including the presence of any outliers, the type of machine learning algorithm chosen, the time efficiency of gradient descent, the facility of finding derivatives and the robustness of the predication. . Generally, however, it is preferable to use a measure which is directly related to the semantics of the problem, i.e. mathematical parameters from the field of statistics and probability. Loss functions can be classified into two types:

- Classification (used in discrete system)
- Regression (used in continues system)

Loss functions can be classified into two types: classification and regression loss. In this work we will focus on regression loss which is akin to our algorithm. The most commonly used regression loss functions are:

- Mean Square Error (MSE) is the sum of the squared distances between the target variable and the predicted values. It is one of the most widely used regression loss functions.

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n} \quad (18)$$

- Mean Absolute Error (MAE) is the sum of the absolute differences between our target and predicted variables. It measures the mean magnitude of the errors in a set of predictions, without considering their preferred directions. With a preferred direction we obtain the Mean Bias Error (MBE), which is a sum of residuals. The range goes from zero to infinity.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n} \quad (19)$$

The quadratic error is easier to solve, but the absolute error is more true when there are outliers in the dataset. Given the algorithm our goal is to find the point that minimises the loss function. The minimum for

both will be reached when the prediction is exactly equal to the true value. Since the MSE squares the error, the value of the error (e) increases greatly if it is greater than one. If we have an outlier in our data, the value will be much higher. This will cause the model with MSE loss to give more weight to outliers, hence for values that are not representative of our dataset, than a model with MAE loss. If on the other hand, we try to adjust MSE to minimise that single outlier case at the expense of the others, this will reduce its overall performance. MAE loss is useful if the training data have unrepresentative values, this is because if we were to provide only one prediction for all observations that attempt to minimise MSE, then that prediction is supposed to be the average of all target values [2]. But if we are trying to minimize the MAE, that prediction would be the median of all observations. We know that the median is more robust to outliers than the mean, which consequently makes MAE more efficient at outliers than MSE. A big problem with using MAE loss, in neural networks, is that its gradient is fixed throughout the training phase, so we will have a large gradient even for small values of loss. This is not good for learning. We will see later to use a dynamic learning rate that decreases as we approach the minima. MSE, on the other hand, will converge well even with a fixed learning rate. If the outliers represent important anomalies or data sets that represent more of our dataset then we should use MSE. On the other hand, if we believe that the outliers represent small clusters, then we should choose MAE as the loss function.

From these considerations and as reported earlier, when we analysed our two datasets, the choice falls on MAE because we are not sure that the outliers obtained, as in the GRi-mech 3.0 dataset, are representative for the study of our algorithm.

2.5.3 Deep neural algorithm

If in the previous chapters we wanted to show an overview of the theory behind this thesis work, introducing the phenomenological part of flame front propagation and combustion in methane engines and a brief description of machine learning, in this chapter we want to start entering the specifics of the actual work done to create our neural network. Starting from a previous study conducted on the correlation that exists between pressure, EGR, temperature, dosage and laminar speed. The aim of this research is to apply and validate a ML machine learning model that is totally unrelated to the mathematical phenomenological aspect of combustion and is able to find correlations between our input data and laminar speed. The present model, as mentioned above, exploits a supervised machine learning algorithm called Linear regression, and is implemented in an open-source integrated development environment in the Python computer language. For this purpose, an open-source machine learning library for this programming language is used, called Scikit-Learn and within which contains the previously mentioned algorithms, and designed to work with other libraries such as NumPy and SciPy. libraries such as NumPy and SciPy that contain most of the functions used in the predictive model code of the predictive model. We will first look at the structure of this code and then at the analysis and results. then move on to analysis and results. In our model, to create our algorithm, we should choose:

- number of hidden layers
- number of neurons per layer
- optimization function
- activation function
- percentage of dropouts

Number of hidden layers

As mentioned above, neural networks had only three types of layers: hidden, input and output. We can see these as all being of the same layer type, if we consider that the input layers are fed by external data (no

previous layers) and the outputs feed the data outwards (final result). These three layers are now called dense layers, because each neuron in this one is *fully connected* to the next layer. Modern neural networks have many other types of layers to deal with. Besides the mentioned dense layers, we also have dropout, convolutional and recurrent layers. Very often dense states are used together with the other types of layers [10]. Our algorithm, as a first attempt, plans to use only dense layers. When considering the structure of dense layers, two fundamental decisions must be made about these hidden layers, how many hidden layers form our neural network and how many neurons will be in each of these layers. Two or fewer layers will often be sufficient with simple data sets, such as one-to-one dependency, however with complex data sets, additional layers may be useful. The general rule that we can apply in our first attempt is [11]:

Number of hidden layers	Goals result
None	to represent separable linear functions or decision functions
1	To approximate functions containing a continuous mapping from one space to another
2	To represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy
> 2	To learn complex representations such as automatic feature design for layers

Table 2.5.3.1: How to choose the number of hidden layers

In our case, considering the input data analysed, we have chosen two densely connected hidden layers, with an output layer that approximates the speed. Let us now see how to choose the number of neurons for each of them.

Number of neurons per layer

Deciding the number of neurons for each hidden layer is very important in deciding the overall architecture of the neural network. These have an enormous influence on the final output, even though they do not interact directly with the outer layer. Therefore, both the number of hidden layers and the number of neurons must be carefully considered. Using too few neurons in the hidden layers will result in *underfitting*, which occurs when there are too few neurons in the hidden layers to adequately detect the connections linking a complicated data set. Conversely, using too many neurons in the hidden layers can lead to *overfitting [2]*. This occurs when the neural network has so much information processing capacity compared to the limited amount of information contained in the training set that it fails to train all the neurons in the hidden layers. However, even if we have an adequate amount of set data, we may have a second error. Indeed, a large number of neurons in the hidden layers exponentially increases the time needed to train the network. Remember that our thesis goal is to create a model that is faster than classical mathematical models, and an increase in computation time is what we do not want. Extending the number of neurons therefore increases the amount of training time, which can increase to such an extent that it will be impossible to train the neural network properly. We will have to make trade-offs between too many and too few neurons in the hidden layers. There are many empirical methods, according to the literature [10], to determine an acceptable number of neurons to be used in the hidden layers:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer
- The number of hidden neurons should be 2/3 of the size of the input layer plus the size of the output layer
- The number of hidden neurons should be less than twice the size of the input layer

These three rules are a starting point to consider, but we must also consider our datasets, they are very different from each other, and we must consider the computational power of our tools. This is why to avoid that the selection of an architecture for the neural network is reduced to trial and error, our model will then be optimized with the tuning of the hyperparameters through *GridSearch*, which optimizes all the factors, important for the creation of the network, and avoid trial and error. Considering the literature and our goals, we choose for the first attempt 64 neurons for each hidden layer.

Optimization function

Training a very large deep neural network can be very slow, to speed up this process and find our weights β, we must use optimization functions that allow us to minimize the delta between predicted value and real value. One of the simplest used is the "*descending gradient*". Gradient descent is a generic optimization algorithm. To reduce the cost function, gradient descent modifies the parameters iteratively. If we think of our training phase, and the reduction of the cost function as a mountain, the strategy for quickly reaching the bottom of it is to go down in the direction of the steeper slope, which is what makes the descent of the gradient, by measuring the gradient to the local error. Another important parameter in the descent of the gradient is the size of the steps, durance the search for the minimum, determined by the rate of learning. If the learning rate is too small, there will be too many iterations and for converging will take a long time. On the contrary, if the learning rate is too high, it may happen that you have an apprenticeship that does not minimize but tries to go up the "mountain", not being able to find a good solution. If we think of minimizing the error as an element in three dimensions, there can be holes, ridges, plateaus and everything types of uneven soils, making convergence at a minimum very difficult [2]. It is important when using gradient descent, you need to make sure that all features have a similar scale, which does not happen in our case, as a result it will take much longer to converge.

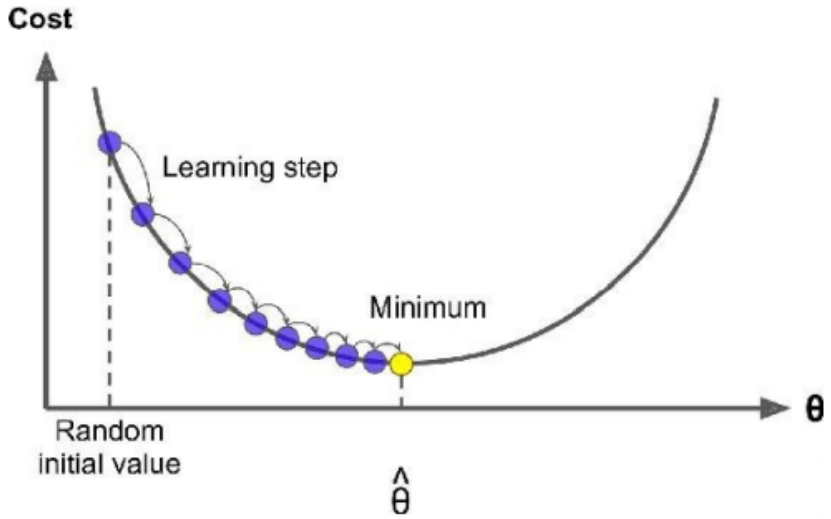

Figure 2.5.3.1: Gradient descent [2]

For that reasons we must watch other optimization function, for our first attempt, that reflecting our aims, i.e. to minimize the learning time with a good result. The problem with gradient descent is to slow down a little too fast and it ends up never converging towards the global optimum. The RMSProp optimization function solves this problem by considering only gradients from the most recent iterations. It focuses on a preferential direction in addition to the vertical minimization of error, considering our error as if it were in 3D, in this way we can have a learning rate faster and a better accuracy. Except on very simple problems, this optimizer almost always works much better than gradient descent also converges faster. We choose for the first attempt RMSProp.

Activation function

The activation function is a "gate" between neurons of two different layers. They decide whether the neuron should be activated or not, otherwise the activation function, weights and bias would simply make a linear transformation, neglecting the connections between the various layers, the (8) show this think.

$$y = activation\ function(\Sigma(weight * input) + bias) \quad (20)$$

Our neural network should have nonlinear activation functions, which can help the network learn complex data. The activation function selects the most important information from the least relevant. Activation functions also have an important effect on the ability of the neural network to converge and the speed of convergence, on the contrary it can happen that activation functions prevent the convergence of neural networks. Our neural network uses nonlinear activation functions, which allow the model to create complex mappings between the inputs and outputs of the network, essential for learning from complex data such as nonlinear data. Using nonlinear activation functions we can represent almost any problem that occurs in the scientific field. Nonlinear functions allow backpropagation because they have a derived function related to inputs and allow stacking multiple layers of neurons to create a deep, multi-layered neural network. In *figure 2.5.3.2* are reported the most popular and used not linear activation function:

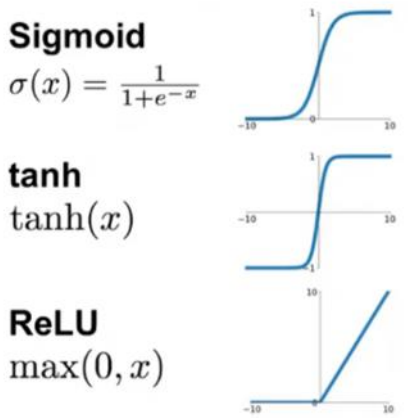

Figure 2.5.3.2: Types of not linear activation functions [1]

The choice of our activation function falls on ReLu, because in our case of analysis we will only have positive input and output data, for this reason it is not necessary to burden our network with activation functions used especially for classification problems such as Sigmoid and Tanh. It is computationally efficient and allows the network to converge very quickly. It is nonlinear, although it looks like a linear function, and allows backpropagation. The fading is done when the inputs are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn, but as said in our case it does not happen.

Percentage of dropouts

The percentage of dropout, indicates the amount of neurons, which are randomly deactivated and therefore do not contribute to learning. It might seem wrong as logic, but when the training process begins they are placed on random values and as the learning progresses they are prevented from making mistakes and changing their weights. But to make sure that the neurons do not depend strictly on a specific weight, so

avoid overfitting, we must select the dropout percentage, which turns them off randomly and lets the other neurons learn and adapt as well. We choose 0.2 for the first attempt value.

Hyperparameter	**value**
Number of hidden layers	2
Number of the neurons per layers	64
Optimization	RMSProp
Activation function	Relu
Percentage of dropout	0.2

Table 2.5.3.2: Standard model review

2.6 Classification network: NaN values

As mentioned above, after analysing our datasets, obtained through simulations using Aramco2.0 and Gri-mech 3.0 using chemical kinetics, we identified NaN values in the column corresponding to the velocity value. All the rows with the pressure, EGR percentage, dosing and temperature values that had these velocity values, as mentioned above, were eliminated before starting the training activity. The need arises to plot these 'non-numbers', showing how many there are for each dataset and for which pressure, EGR percentage, metering and temperature values they occur. Everything therefore consists of architecting, upstream of our main neural network, a non-pass classification network that returns "one" when a non NaN value is present and "zero" when it detects a non-number. Then a counter will be inserted in series which, reading the values obtained, returns a percentage index of the presence of the NaN values with respect to the analysed columns. We must also specify that the Aramco 2.0 dataset has non-speed numbers expressed through the string "NaN", while the Gri-mech 3.0 dataset has the value "1.0000000E-03" in the non-speed number string. For this reason, a network will be built in python to recognise "NaN" values and "1.000000000E-03" values in the speed string or column.

Gri-mech 3.0 dataset result

Temperature [K]	Pressure [bar]	NaN counter		% NaN on total
300	1	53	542	20.63
	5	57		
	10	56		
	20	59		
	40	57		
	80	68		
	120	65		
	160	64		
	200	63		
400	1	52	488	18.57
	5	58		
	10	49		
	20	48		
	40	54		
	80	58		
	120	59		
	160	53		
	200	57		
500	1	43	428	16.29
	5	47		
	10	56		
	20	47		
	40	45		
	80	48		
	120	47		
	160	48		
	200	47		

Temperature [K]	Pressure [bar]	NaN counter	Total NaN	% NaN on total
600	1	37	354	13.47
	5	35		
	10	36		
	20	48		
	40	40		
	80	36		
	120	42		
	160	42		
	200	38		
700	1	39	314	11.95
	5	36		
	10	37		
	20	34		
	40	33		
	80	32		
	120	35		
	160	34		
	200	34		
800	1	40	342	13.01
	5	35		
	10	38		
	20	39		
	40	37		
	80	40		
	120	36		
	160	38		
	200	39		
900	1	39	296	11.26
	5	20		
	10	19		
	20	19		
	40	33		
	80	42		
	120	45		
	160	41		
	200	38		
1000	1	33	302	11.49
	5	32		
	10	35		
	20	33		
	40	34		
	80	33		
	120	35		
	160	33		
	200	34		

Temperature [K]	Pressure [bar]	NaN counter	Total NaN	% NaN on total
1100	1	33	320	12.18
	5	32		
	10	32		
	20	33		
	40	34		
	80	39		
	120	37		
	160	41		
	200	39		
1200	1	35	377	14.35
	5	39		
	10	37		
	20	37		
	40	38		
	80	37		
	120	39		
	160	38		
	200	37		
1600	1	36	322	12.25
	5	35		
	10	36		
	20	37		
	40	36		
	80	35		
	120	36		
	160	35		
	200	36		
Total dataset NaN			2627	100

Table 2.6.1: Table showing the NaN in Gri-mech 3.0 dataset

We can notice how our dataset, initially divided in files collected by temperature and pressure range, show a greater number of non-numbers where the temperature is lower, remembering that the steps of dosing and portion of EGR opening remain equal for each file, respecting the steps reported in table 2.3.2.1. It is to specify that some simulated "limit" points, result in all the files with NaN values of speed, these points present:

- dosing value equal: 0 / 0.2 / 5
- EGR open to 100%

These results were to be expected in a deliberately extreme dataset. We also noted the presence of non-numbers, with stoichiometric ratios equal to 0.3 and 0.4, in a range between 300 K and 500 K. We can see a high number of non-numbers in a very large dataset.

Aramco 2.0 dataset result

EGR [%]	Temperature [K]	NaN counter	Total NaN	% NaN on total
0	300	6	6	4.68
	400	-		
	500	-		
	600	-		
	700	-		
	800	-		
	900	-		
	1000	-		
	1100	-		
10	300	12	12	9.37
	400	-		
	500	-		
	600	-		
	700	-		
	800	-		
	900	-		
	1000	-		
	1100	-		
20	300	8	24	18.75
	400	2		
	500	9		
	600	5		
	700	-		
	800	-		
	900	-		
	1000	-		
	1100	-		
30	300	2	5	3.90
	400	-		
	500	-		
	600	3		
	700	-		
	800	-		
	900	-		
	1000	-		
	1100	-		
40	300	20	28	21.87
	400	8		
	500	-		
	600	-		
	700	-		
	800	-		
	900	-		
	1000	-		
	1100	-		

EGR [%]	Temperature [K]	NaN counter	Total NaN	% NaN on total
50	300	24	53	44.53
	400	22		
	500	19		
	600	-		
	700	-		
	800	-		
	900	-		
	1000	4		
	1100	8		
Total dataset NaN			128	100

Table 2.6.2: Table showing the NaN in Aramco 2.0 dataset

As we can see, our Aramco dataset is derived from files saved by EGR value, so as the first column on the left we find the EGR opening percentage. We can see that in this dataset most of the "non-numbers" are concentrated on a temperature value equal to 300 K. In fact, if we consider a dosing value for these simulations always equal to one, and thirty-three pressure steps, we notice that the NaN values found at 300 K are considerable. Moreover, we notice that the non-numbers increase as the percentage of EGR opening increases, and at the same time the temperature values where we find these values increase.

We note that the temperature range between seven hundred and nine hundred Kelvin is almost free of non-numbers, and also that almost 50% of these values were found with an EGR value equal to 50%. This is very significant if we look critically at the Gri-mech dataset, where excessively high velocity values were found near the same EGR value. We will see later that this will affect the learning of our model (which uses the Gri-mech dataset), and how our optimisation will be less efficient.

3 First results

Before presenting the first results we must introduce the "*Loss value"*, this is to indicate a sum of the errors made for each example in the training or validation sets. The loss value implies how good or bad a given model behaves after each optimization iteration. Ideally, you would expect loss reduction after each or more iterations. The other variable to present is the number of "*epochs*", it is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset. An epoch means that each sample in the training dataset had the opportunity to update the parameters of the internal model. Our model was initially trained on 1000 epochs. The latter is a high value that leads to a slowdown in the learning phase of our model, but it is necessary to understand how this behaves and how we can improve and speed it up.

3.1 First attempt Aramco 2.0 results

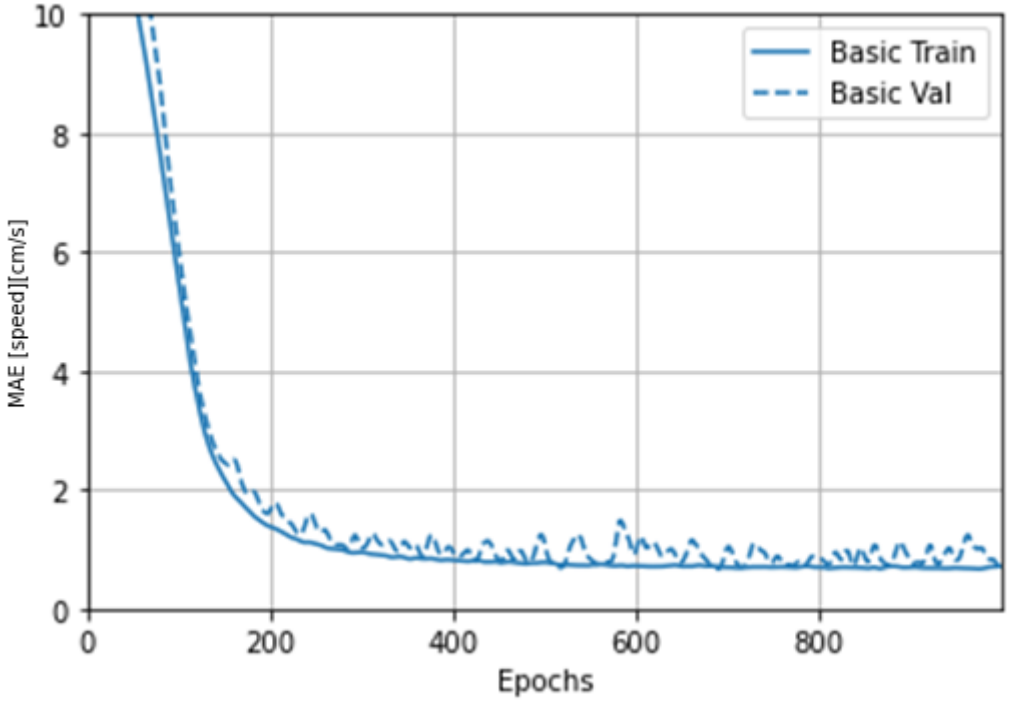

Figure 3.1.1: MAE trend over 1000 epochs with standard model

We can see how our model in the range of 200 epochs, meets our expectations, with a high learning rate, and with a difference, between the estimated target value of the model and the actual value, very close up to 200 epochs. However, when you exceed this threshold you do not get to minimize our objective function, which remains close to the value of one. We also notice that going beyond 200 epochs, only slows down our algorithm, as this is evident from the spikes that increase, going towards the plateau.

From the figure 3.1.1, we can see how a very high learning rate, leads to a worsening of the learning phase, around the 200 epochs. To improve the perform of our model we must limit the oscillations and avoid the unnecessary learning phase time, through a cut to the number of epochs for our training phase.

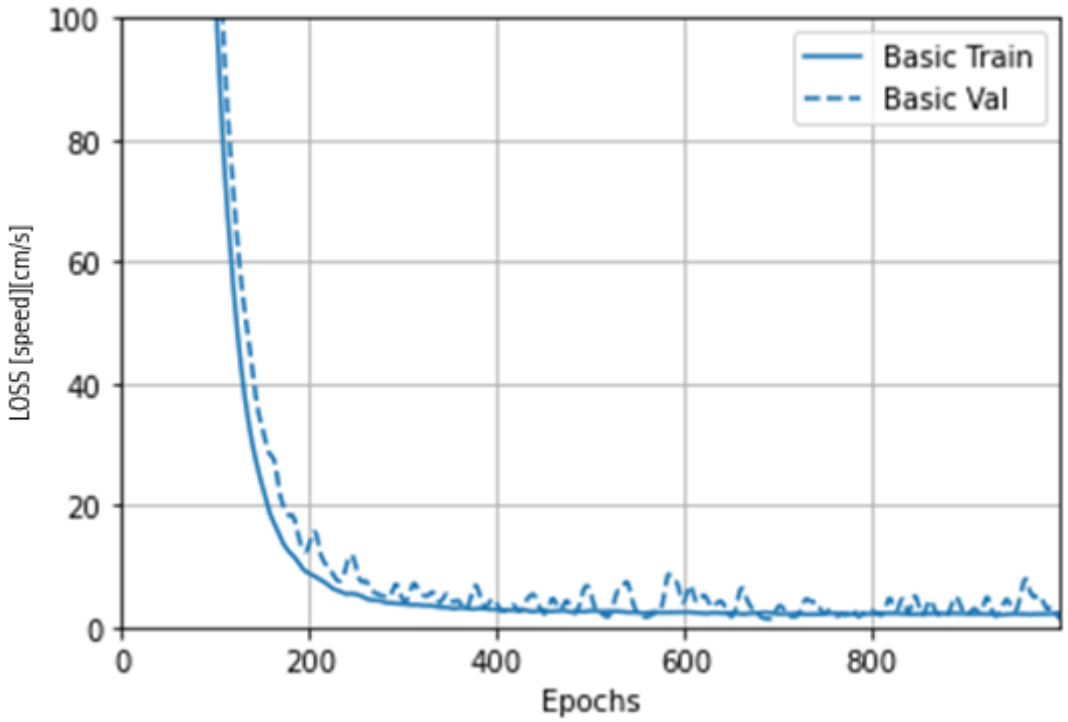

Figure 3.1.2: LOSS trend over 1000 epochs with standard model

Finding a good learning rate can be difficult, in fact if you set it too high, the training may diverge. On the contrary, if you set it too low, the training will eventually converge to the optimal, but it will take a lot of time. If the learning will be just above the optimal value, it will progress very quickly at first, but will end to drift around the optimal value, without ever reaching the plateau. In our case we used an adaptive learning rate, using as an activation function RMSProp, but even in this case it may take some time to stabilize. In addition, if you have a limited calculation budget, you will need to stop training before it converges properly, resulting in a non-optimal solution.

From Figures 3.1.1 and 3.1.2, it is noted to increase the speed of our model, we have to stop learning the algorithm on 200 epochs, this to avoid obvious overfitting problems. The ways to avoid overfitting are multiple:

- *Retraining the neural networks* using the same model, the same training set but with different weights, and then choose the network with the best performance
- *Multiple neural networks* that see multiple neural networks trained in parallel, same structure but different weights, and at the end make an average of their output for choose the best
- *Regularization* that for decrease the bias and weighs, adding a term to the error function, witch as result a smooth outputs with a low tendency to overfitting
- *Tuning performance ratio* that is similar to regularization but use a particular parameter that indicated how much the network need to regularized
- *Early stopping* that monitoring the error (LOSS) behaviour after each iteration, and stopping the training when overfitting starts.

Obviously we will use something easy to implement with phyton code, where we automatically stop training when the validation result does not improve. We will then use the EarlyStopping callback function that checks the training condition at each epoch. If a given number of epochs pass without showing improvement, then training is automatically stopped. The results of applying the function are shown in the figures 3.1.3 and 3.1.4 and reveal an improvement in both computation time and avoidance of overfitting.

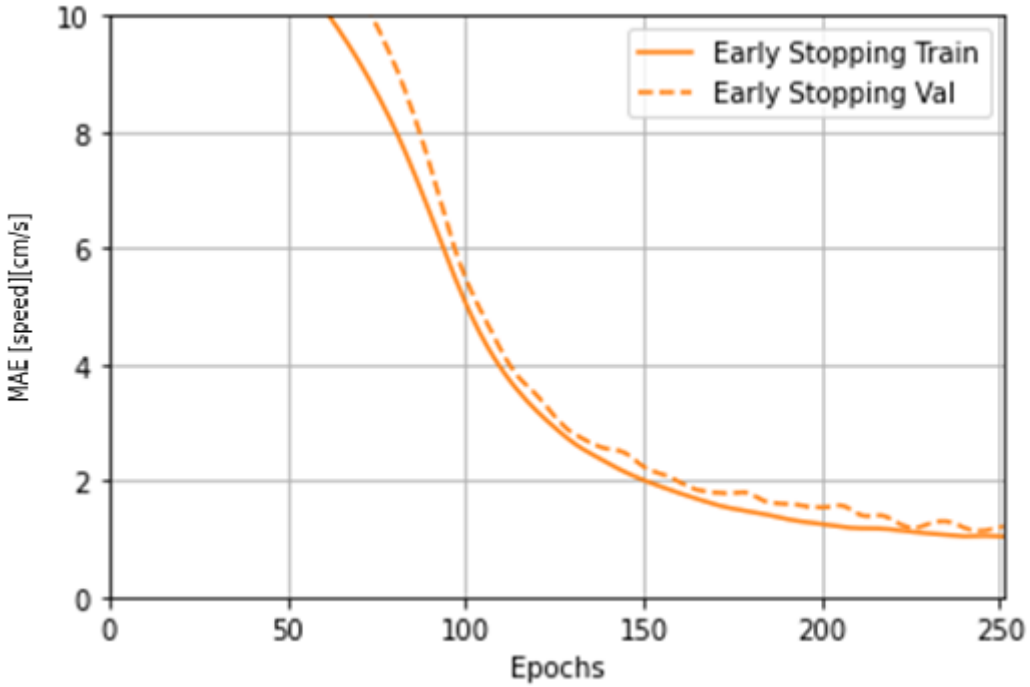

Figure 3.1.3: MAE trend over 250 epochs with standard model

The graph shows that the error on the validation set is usually around +/- 1 [cm/s]. A satisfactory result, if we consider that our neural network has not been optimised now. It should also be noted that there is still margin for improvement, because there are still small oscillations around 200 epochs.

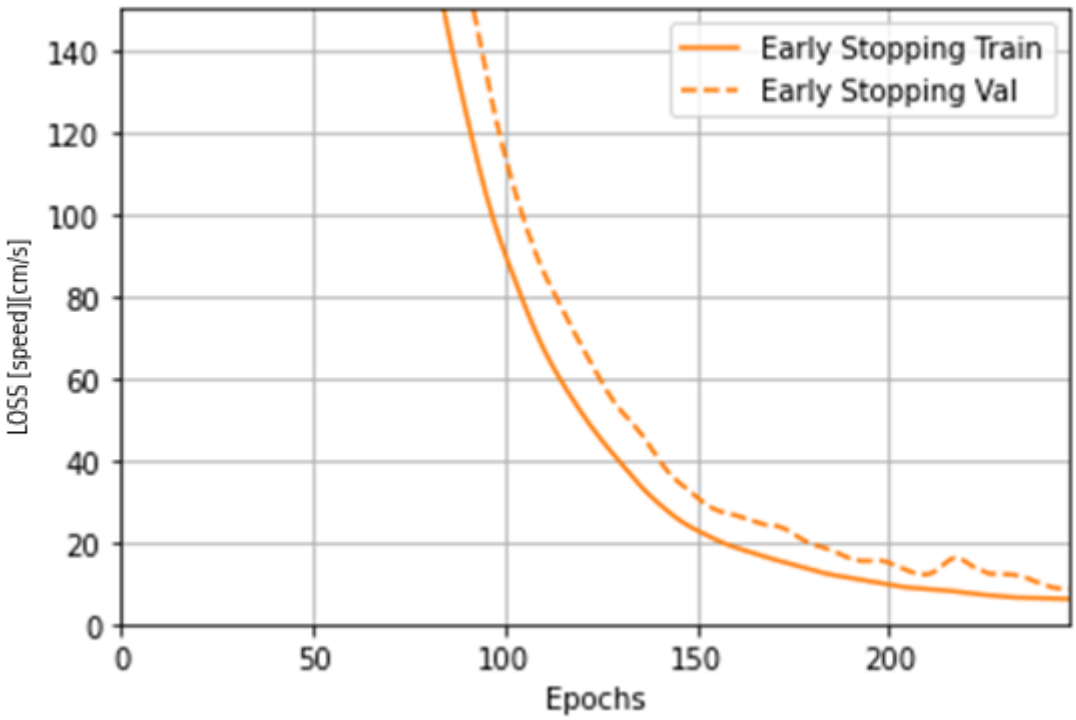

Figure 3.1.4: LOSS trend over 250 epochs with standard model

Loss value is also decreasing over the 200 epochs. Both training loss and validation loss decrease exponentially as the number of epochs increases, suggesting that the model acquires a high degree of accuracy as our epochs (or number of forward and backward steps) increase. Again, there is room for improvement.

Having seen how our algorithm performs in the training phase, it is now necessary to verify its performance when using our test dataset, remembering that it was not used for the learning and validation phase. Assess the hidden non-linear correlations from our training dataset and evaluate them on our test set. The graph below presents the speed obtained from the test set on the x-axis and on the y-axis the speed predicted by the training.

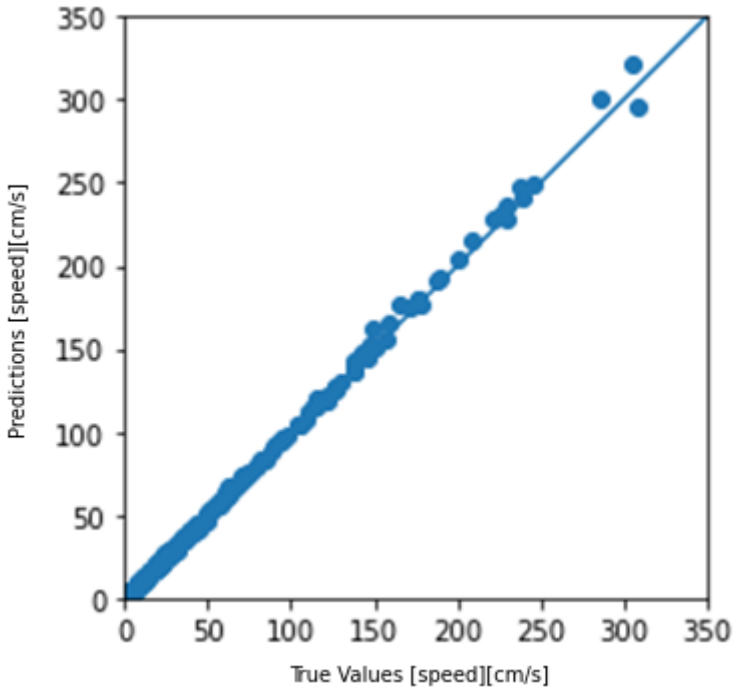

Figure 3.1.5: Results obtained by testing our standard algorithm

Our model seems to predict well, as evidenced by the many points that fit our regression line well. This shows that we can use deep learning to understand the non-linear relationships of the laminar velocity phenomenon in methane air mixtures. Now let us see from figure 3.1.6 how our error is distributed between actual and predicted value:

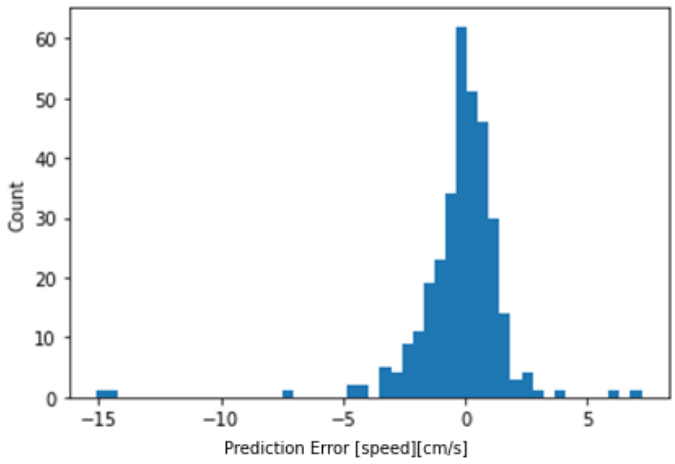

Figure 3.1.6: Error distribution obtained by testing our standard algorithm

The distribution of our error is similar to a Gaussian. We see that our model fits quite well and does not have high error values between predicted and actual values. There is a larger discrepancy around high velocity values, with a difference that however remains in the range of 10-20 cm/s. In general, we have an error between predicted and actual values that is always below 5 cm/s.

3.2 First attempt GRI-Mech 3.0 results

It will be very important for our model to see how the same algorithm behaves with a dataset, which as pointed out above is totally different from the one analysed previously (Aramco 2.0). We would like to mention that we have not changed the hyperparameters, which will be optimised for both datasets later. The goal is to see if with the Gri-mech dataset, the functions to evaluate learning and testing will change, and by how much.

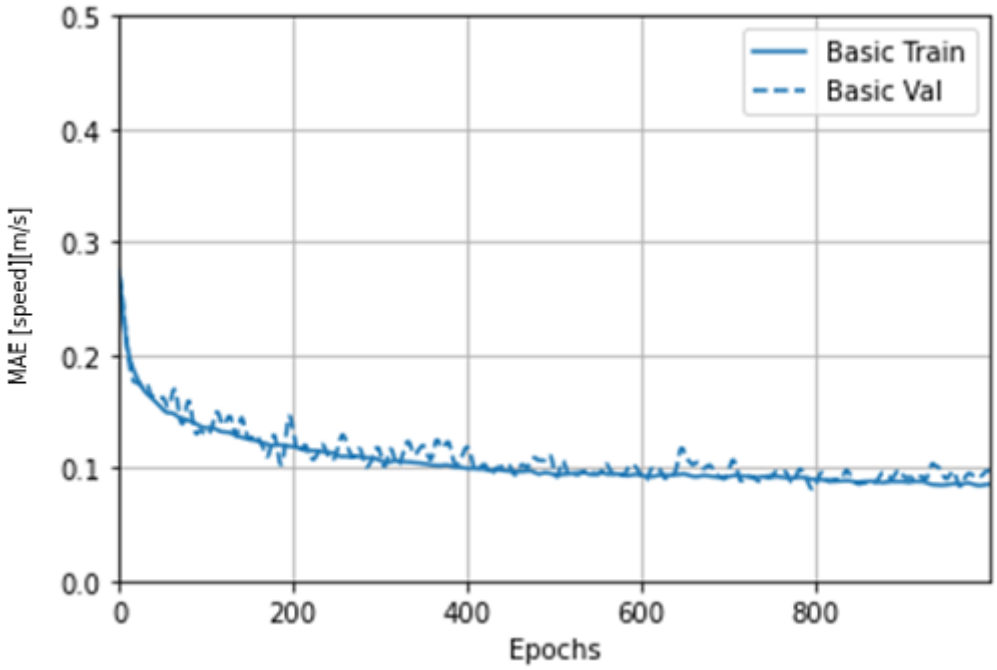

Figure 3.2.1: MAE trend over 1000 epochs with standard model

As it is clear, the MAE, i.e. the difference between the estimated target value of the model and the actual value is much smaller than before, but this should not deceive us because our data obtained from the GRI-mech dataset are expressed in m/s. We notice as a big difference that the MAE in this case starts its learning phase with a value almost comparable to the previously analysed case, but the trend towards the plateau is not correct. We notice that there is a sudden decrease in the MAE value and then it settles on values of 0.1 m/s or 10 cm/s. This is a first indication that our "standard" algorithm does not perfectly fit the starting dataset. In fact, as analysed in the previous chapters, although it is very different, it is almost eight times larger than the Aramco 2.0 dataset. As a result we have too high a learning rate in the initial phase which leads our learning curve to diverge. We now go on to analyse the loss function.

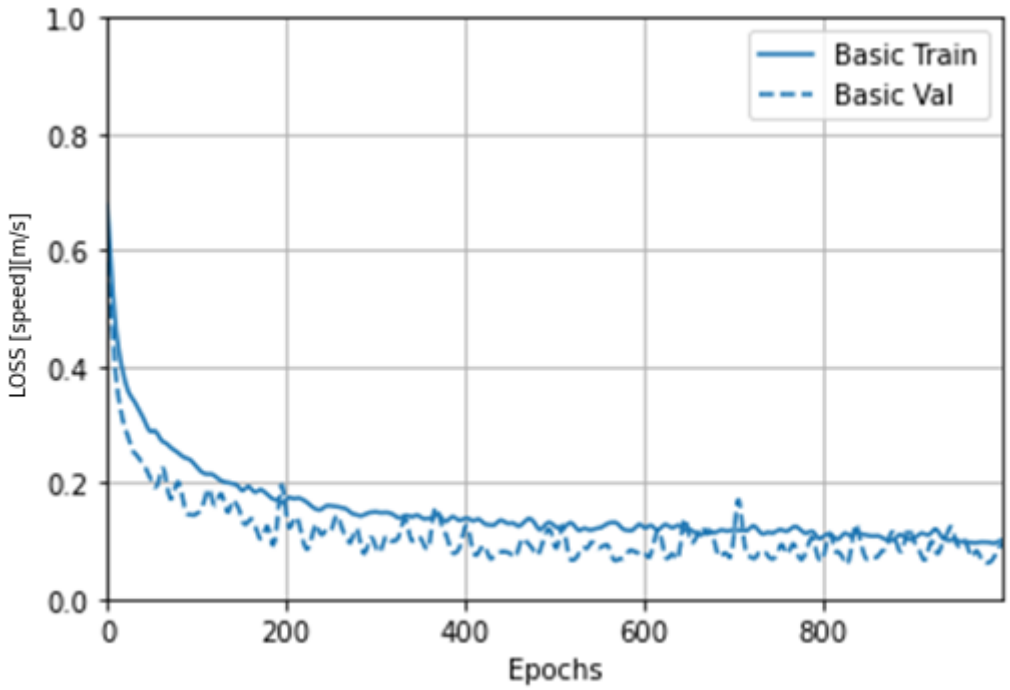

Figure 3.2.2: LOSS trend over 1000 epochs with standard model

Also in this case we notice that the scales compared to the case previously analysed are totally different. We note that the learning rate is not high, probably it will converge to the target value with a large number of epochs. In this case we note the presence of a marked underfitting. Underfitting occurs when the neural network is not able to learn accurately from the training set, and it will be even more evident how well our model will not be able to fit the validation set. This is characterised by high bias and high variance. How to avoid underfitting in a neural network:

- *Add layers of neurons or increase inputs* this can generate more complex predictions and improve model fit
- *Add more training samples and improve their quality*, this way the more training samples fit into the network and the better they represent the data population, as consequence we will have a better behaviour from our network
- *Increase the dropout percentage*, has as consequence the increasing neurons number deactivated in each training iteration ensuring that some learned information is removed randomly, also reducing the risk of overfitting
- *Decrease the regularization parameter* using a regularization performance parameter to set the optimal degree of regularization, which can help the model to fit better.

The reduction of overfitting will be done later when we go to optimise the hyperparameters, in an automatic way, through a piece of code that will be presented in the next part. This is to avoid trial and error and to automate the optimisation process of our deep learning algorithm. We will again use the function already implemented in our code, which aims to reduce overfitting.

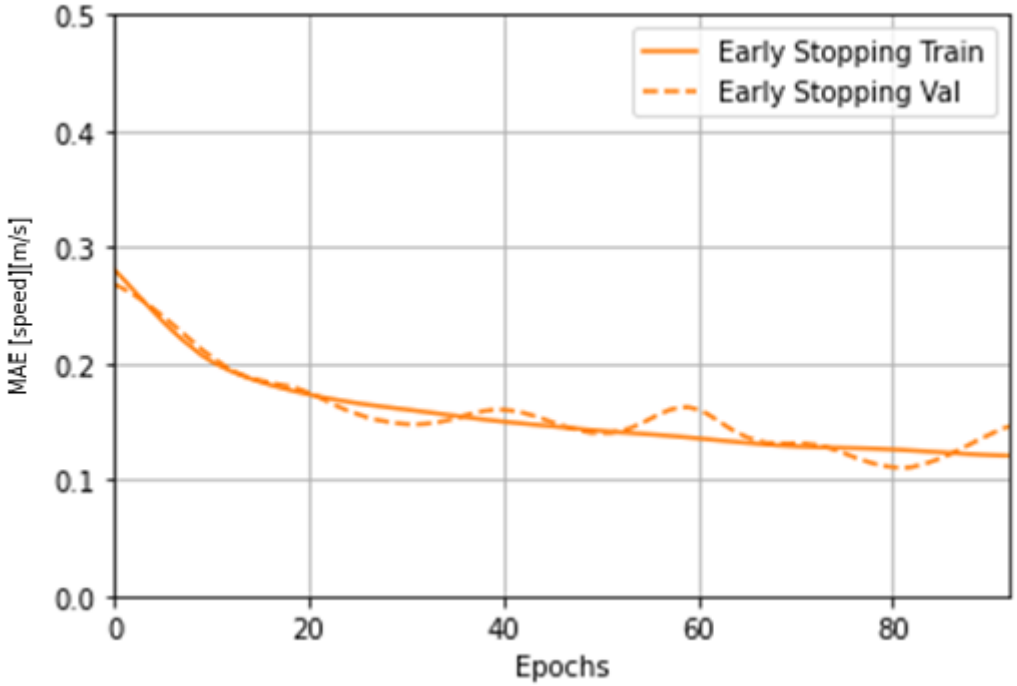

Figure 3.2.3: MAE trend over 100 epochs with standard model

We can see that our early stopping function cuts the learning state to about 100 epochs, which is not a satisfactory result, but expected, given the high value of MAE and LOSS. We can also see a slight presence of underfitting and overfitting. Even having set a function that limits overfitting, in this case we need to optimise the hyper parameters to achieve more satisfactory results.

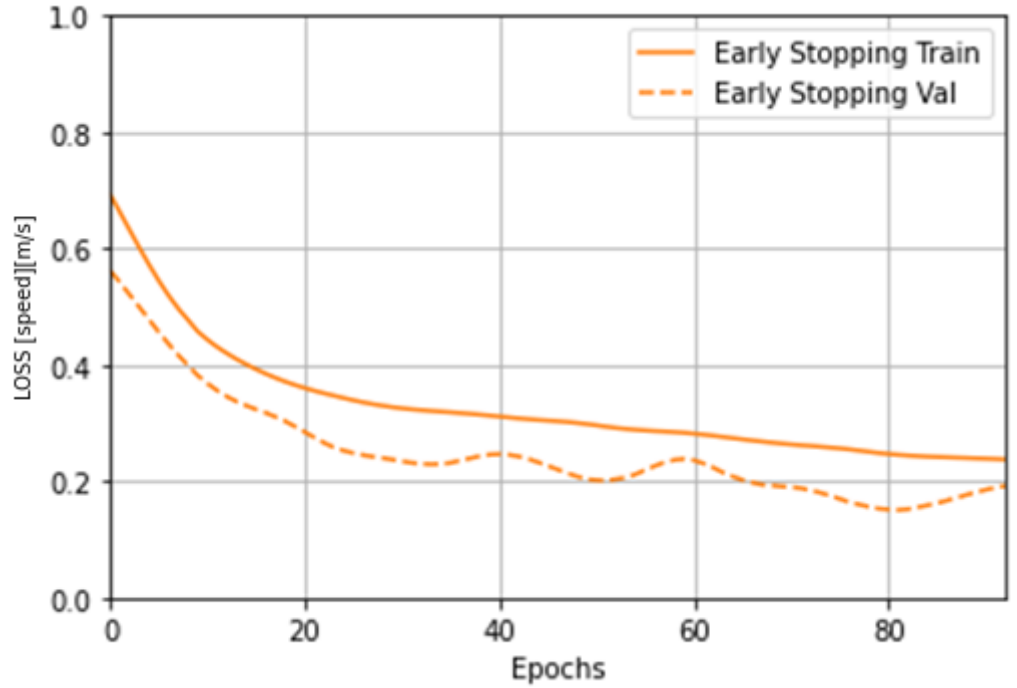

Figure 3.2.5: LOSS trend over 100 epochs with standard model

Even in the case of the LOSS function we are not satisfied with the trend shown, so optimisation using hyperparameters will be very important. Again, we use the test dataset to test our network and to see if it has found and created the right correlations through the deep learning algorithm used.

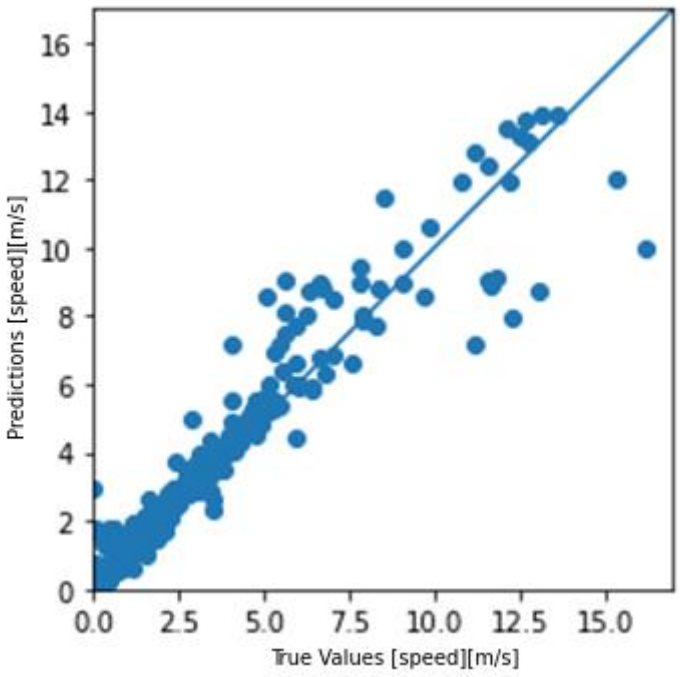

Figure 3.2.5: Results obtain by testing our standard algorithm

It is possible to see how in this case the testing of our network is strongly influenced by the marked underfitting, noted above. on both high and low speed values, we notice a marked difference between predicted and actual values, which is also reflected in the error distribution.

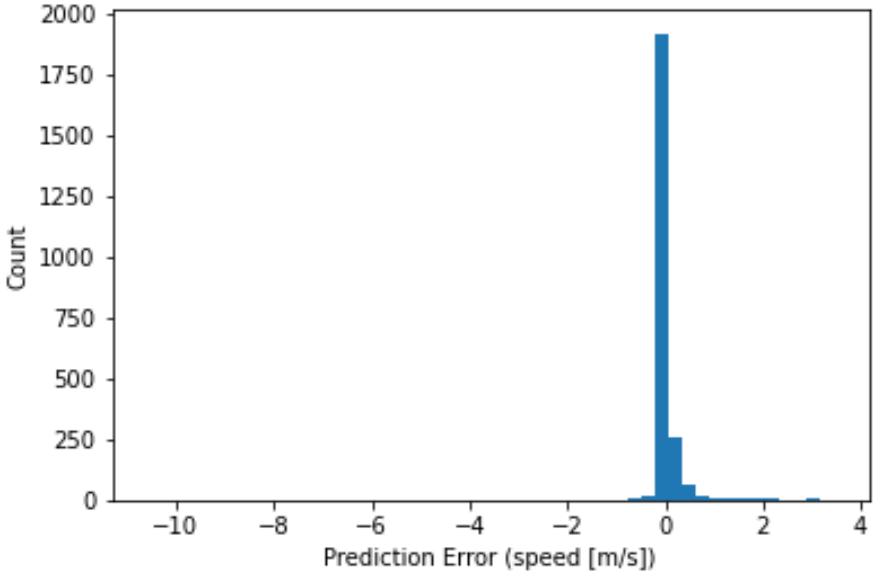

Figure 3.2.6: Error distribution obtain by testing our standard algorithm

We can see how the error distribution includes values that also reach a delta of 4 m/s. This confirms a non-optimal optimization of our model. This was predictable using the same model used for a completely different dataset

3.3 Methods for hyperparameter optimisation

Hyperparameters determine the behaviour of our neural structure, how it adapts to different input values and how it works with the different elements that compose it. A hyperparameter is an *external parameter* set by the neural network operator, as opposed to a parameter that is part of the network learning and cannot be changed. It may seem strange but let us try to give some examples, such as the number of training iterations, the number of hidden layers, or the activation function, these are all hyperparameters, by changing them we can radically change our network. In a neural network, we try many possible values of hyperparameters and see which one works best. There are four main methods for optimising hyperparameters [4]:

- *Manual tuning of hyperparameters*: this method is time consuming and therefore inefficient for fast optimisation, as well as requiring trial and error
- *Grid search*: this involves systematically testing multiple values of each hyperparameter, using lines of code that we can think of as many cascading for-cycles, and then retraining the model for each combination
- *Random search*: a more effective method than manual optimisation
- *Bayesian optimisation*: a method that trains the model several times with different values of hyperparameters, observing the shape of the function generated by these, extending the best function to the whole model.

In our work we chose to optimize the hyper parameters using the Grid search, because it is easy to implement in phyton and does not require a powerful GPU. To do this we have two options in phyton, use scikit-learn method, one is GridSearchCV which uses the scikit-learn API wrapper in Keras, the other method is Hyperas which allows us to train models faster and is also an easy way to approach hyper parameter tuning. Our choice falls on *Hypera* as it is a faster way of optimisation.

3.3.1 Grid search optimised hyperparameters: results with Aramco 2.0 dataset

The following tables give an account of what the hyperparameters are before and after grid search optimisation.

Hyperparameters	**value**
Number of hidden layers	2
Number of the neurons per layers	64-64
Optimization	RMSProp
Learning rate	0.01
Activation function	Relu
Percentage of dropout	0.2

Table 3.3.1.1: Aramco 2.0 standard model review

We see how the number of neurons per hidden layer has changed, with a decrease that allows us to resolve the problems encountered with overfitting. In addition, the type of optimiser has also changed, i.e. the way in which we try to reduce the loss between the target value and the real value.

Optimised hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	64-32
Optimization	Adam
Learning rate	0.01
Activation function	Relu
Percentage of dropout	0.2

Table 3.3.1.2: Model Aramco 2.0 optimised using Grid search

Adam is an optimisation algorithm that can be used in place of stochastic gradient descent to update network weights iteratively based on training. Stochastic gradient descent maintains a single learning rate and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and adapted separately during learning, and the algorithm calculates individual adaptive learning rates for the different parameters from the first and second moment estimates of the gradients. Instead of adjusting parameter learning rates based on the mean of the first moment as in RMSProp, Adam also makes use of the mean of the second moments of the gradients (the uncentred variance). In particular, the algorithm computing an exponential moving average of the gradient and the squared gradient, checking the decay rates of these moving averages. We can see below what results have been achieved compared to those analysed in the previous chapter.

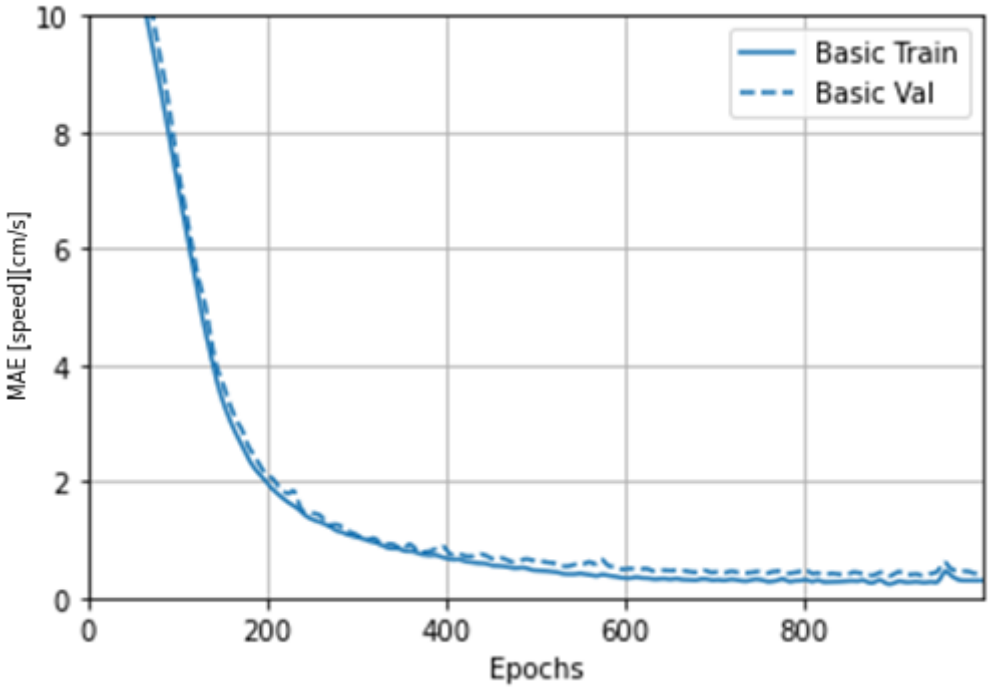

Figure 3.3.1.1: MAE trend over 1000 epochs with Grid Search optimization

Unlike the previous case, i.e. from figure 3.3.1.1, we do not see a marked oscillation, once the 200 epochs have been reached, in fact we have a more linear trend, with a difference between estimated and actual plate value that is less marked even as the number of epochs increases.

Also in this case, the LOSS curve in figure 3.3.1.2, which shows us how our system is learning for each iteration, shows an almost complete disappearance of overfitting, with the loss value decreasing, as we would expect from learning.

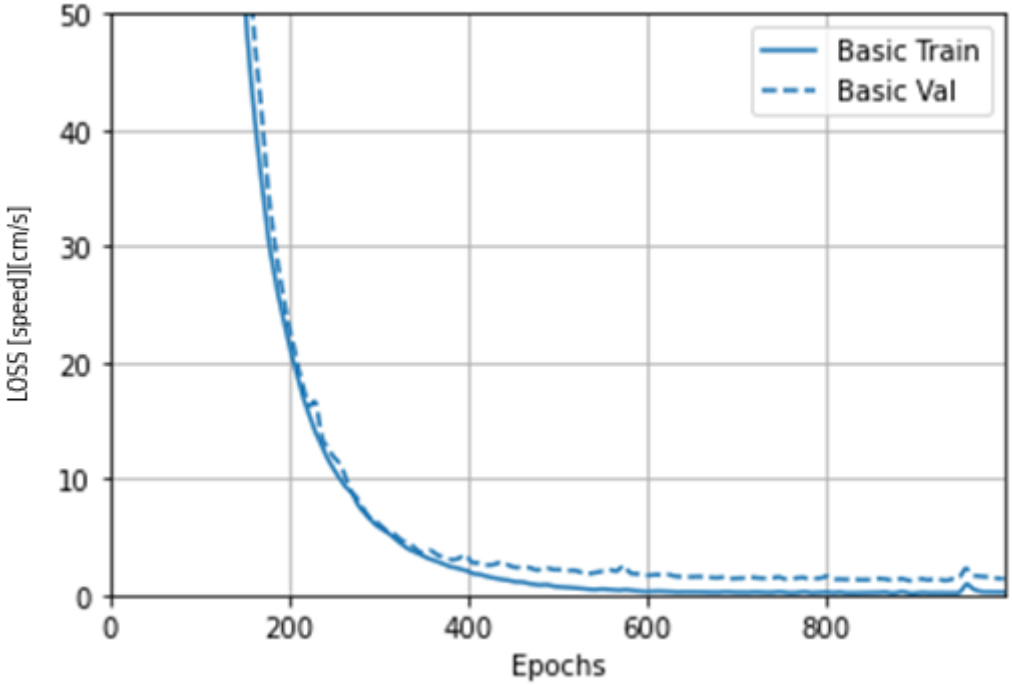

Figure 3.3.1.2: LOSS trend over 1000 epochs with Grid Search optimization

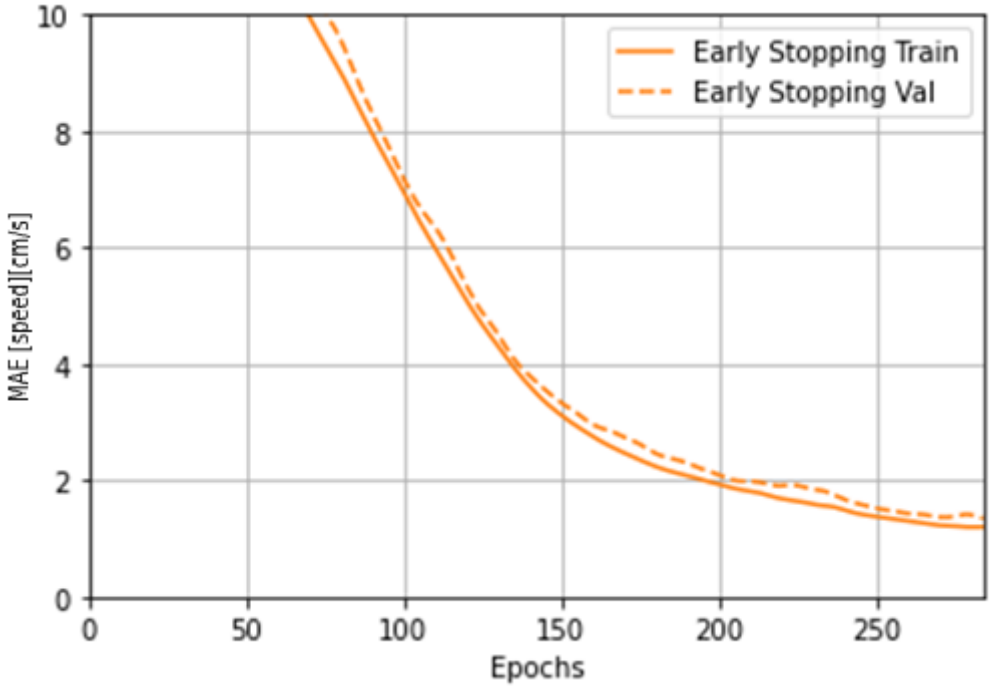

Figure 3.2.1.3: MAE trend over 250 epochs with Grid Search optimization

The use of the Early stopping function helps our model to be faster by stopping the training phase when there is no clear improvement set as a target in the code. The differences do not seem to be substantial, the overfitting has been slightly reduced. The graphs show that the error on the validation set is around +/- 1 [cm/s]. A satisfactory result. We will see later how we have more marked advantages in the loss graph.

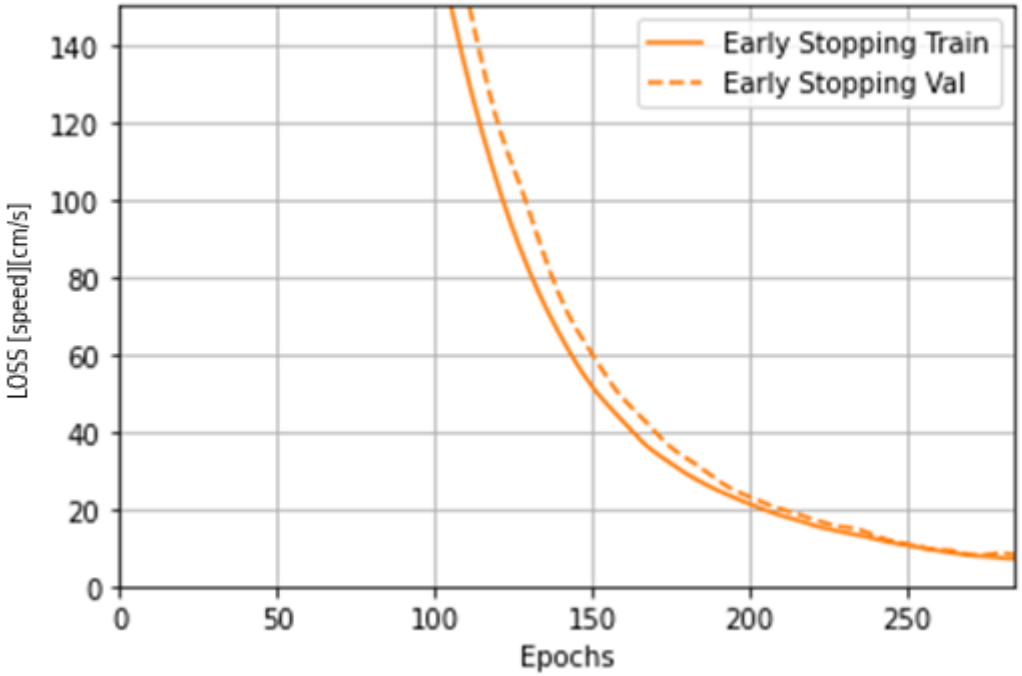

Figure 3.2.1.4: LOSS trend over 250 epochs with Grid Search optimization

Loss value is also decreasing over the 250 epochs. Both the training loss and the validation loss decrease exponentially as the number of epochs increases, suggesting that the model acquires a high degree of precision as our epochs increase. The differences in this case, with respect to the same graph seen in the previous chapters, are substantial, in fact the two curves tend to join, a sign that our model tends to make less errors with the passing of the epochs. Final LOSS reduced by 50% with an optimal delta at the end of training.

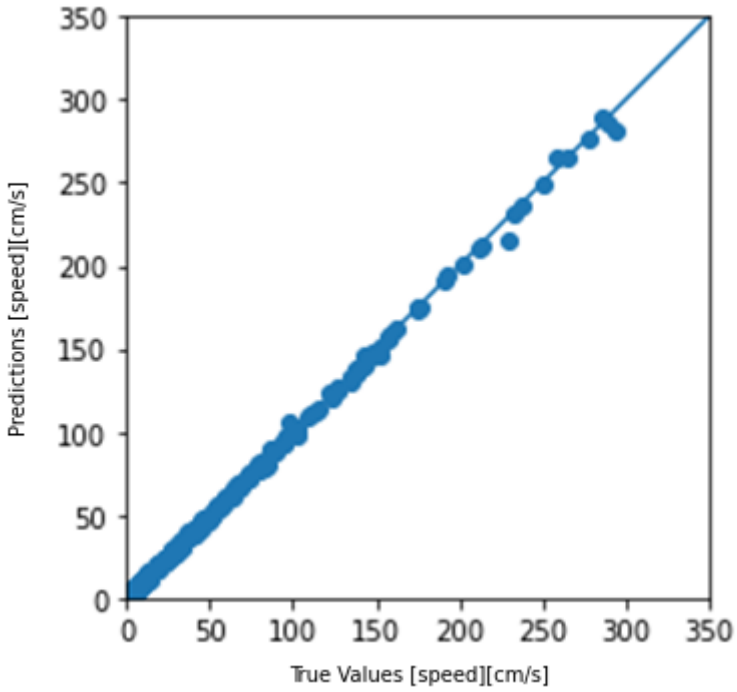

Figure 3.2.1.5: Results obtained by testing our optimised algorithm using Grid Search

It is time to use our test set, remember it has never been "seen" by our model, to evaluate the performance of our model. Our model seems to predict well. There are two differences with respect to what has been analysed in the previous chapters: a better fitting in the part with high velocity values and a slight worsening in the central part of the values. We can see below, that the error distribution also replicates a Gaussian trend and the "prediction" error remains below the range of +/- 5 cm/s. A smaller error that increases for a few medium speed values, but a more appreciable result compared to the standard algorithm.

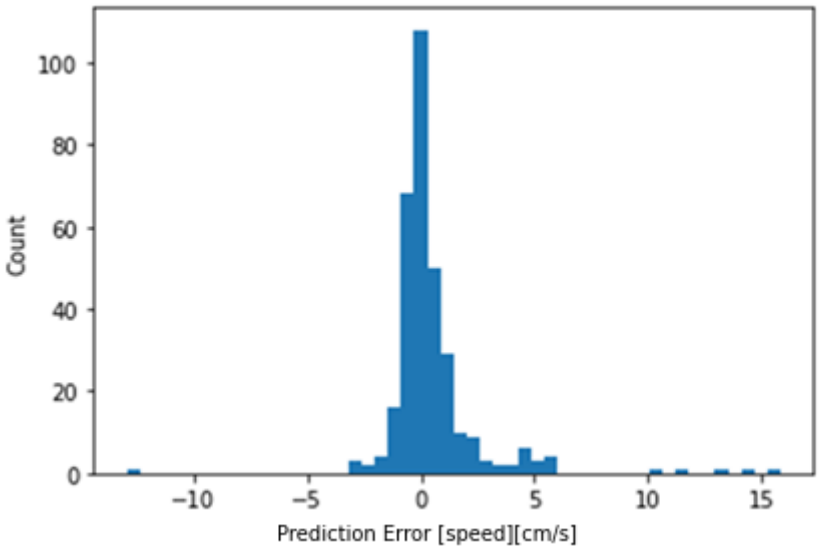

Figure 3.2.1.6: Error distribution obtained by testing our optimised algorithm using Grid Search

3.3.2 Grid search optimised hyperparameters: results with Gri-mech 3.0 dataset

We move on to our second dataset, analysing how our algorithm changes between the optimised and non-optimised model.

Hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	64-64
Optimization	RMSProp
Learning rate	0.01
Activation function	Relu
Percentage of dropout	0.2

Table 3.3.2.1: Gri-mech 3.0 standard model review

The differences we notice concern the number of neurons per hidden layer, an increase of which is justified by the underfitting problems we previously had with the standard model. In fact, we must remember that the Grimech 3.0 dataset is about eight times larger than the Aramco 2.0 dataset, hence the need to increase the number of neurons to explore and to increase the learning capacity of our system. We also see how our leraning rate has changed, which we can justify as a natural consequence of an increase in the number of neurons. In fact, if we increase the capacity of learning, we must limit a descent of the loss gradient that is too fast, which would lead to a divergence between the targhet value and the current value. We also see that the optimiser has not changed, even if it is adaptive to the learning process as said before, and the percentage of dropout in the two levels remains unchanged.

Optimised hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	128-64
Optimization	RMSprop
Learning rate	0.001
Activation function	Relu
Percentage of dropout	0.2

Table 3.3.2.2: Gri-mech 3.0 model optimised using Grid search

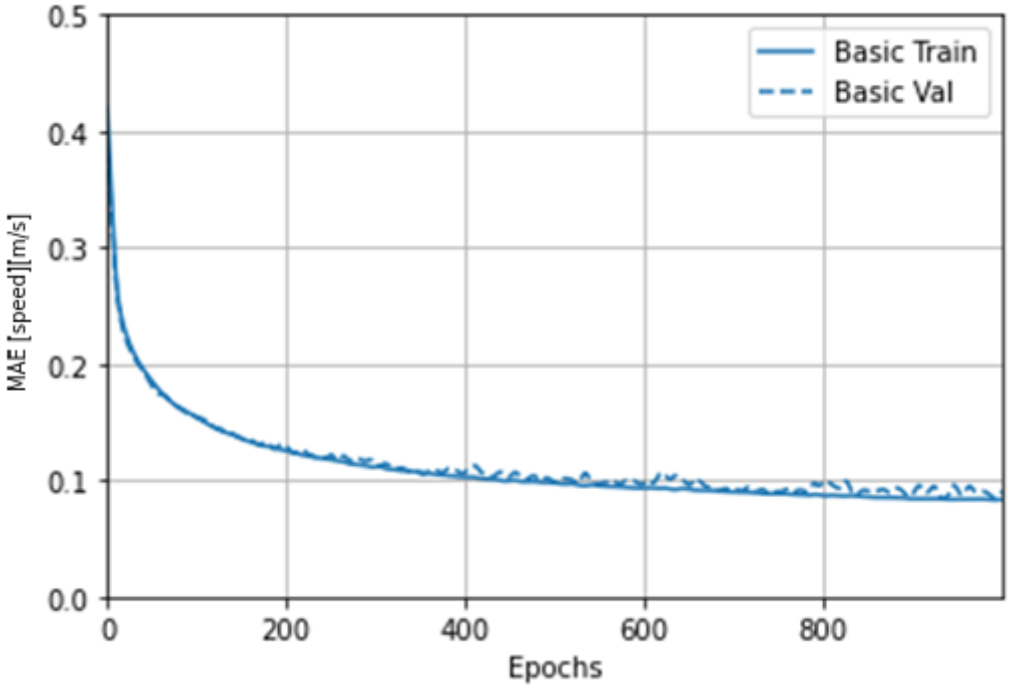

Figure 3.3.2.1: MAE trend over 1000 epochs with Grid Search optimization

The graph depicting the MAE trend over a thousand epochs, *figure 3.3.2.1,* has a behaviour that shows less overfitting, compared to the same graph analysed in the previous chapters with Grimech 3.0 dataset. This result is also obtained by increasing the number of neurons for the hidden layers, this shows us that our optimisation through Grid search has been successful. We see that there is still a slight overfitting from the four hundred epochs onwards, but we will see later how to improve this behaviour using Grid Search and Cross Validation. We always notice within the four hundred epochs that our target value is quite close to our actual value.

This is even more evident with the LOSS graph, *figure 3.3.2.2* which shows us how our model learns during each iteration. In fact, with respect to the same graph of the loss value in the non-optimised model, we see how the underfitting has almost completely disappeared and the training and validation learning curves, which come together around four hundred epochs show how our algorithm learns better and reaches a lower loss value which will be more evident when we analyse the graph of the loss value with early stopping function.

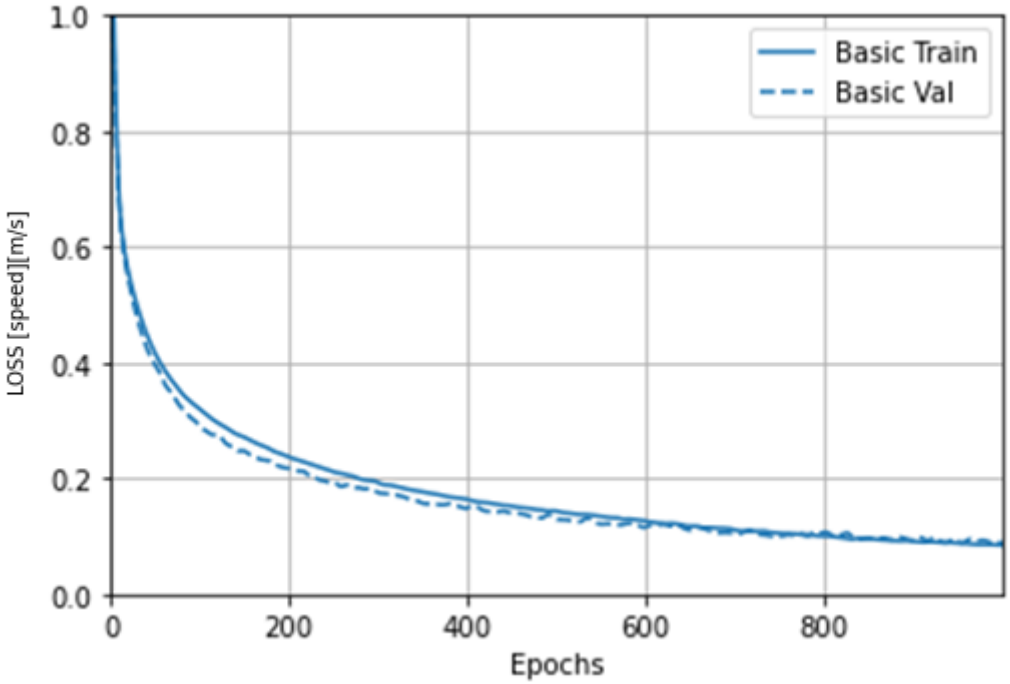

Figure 3.2.2.2: LOSS trend over 1000 epochs with Grid Search optimization

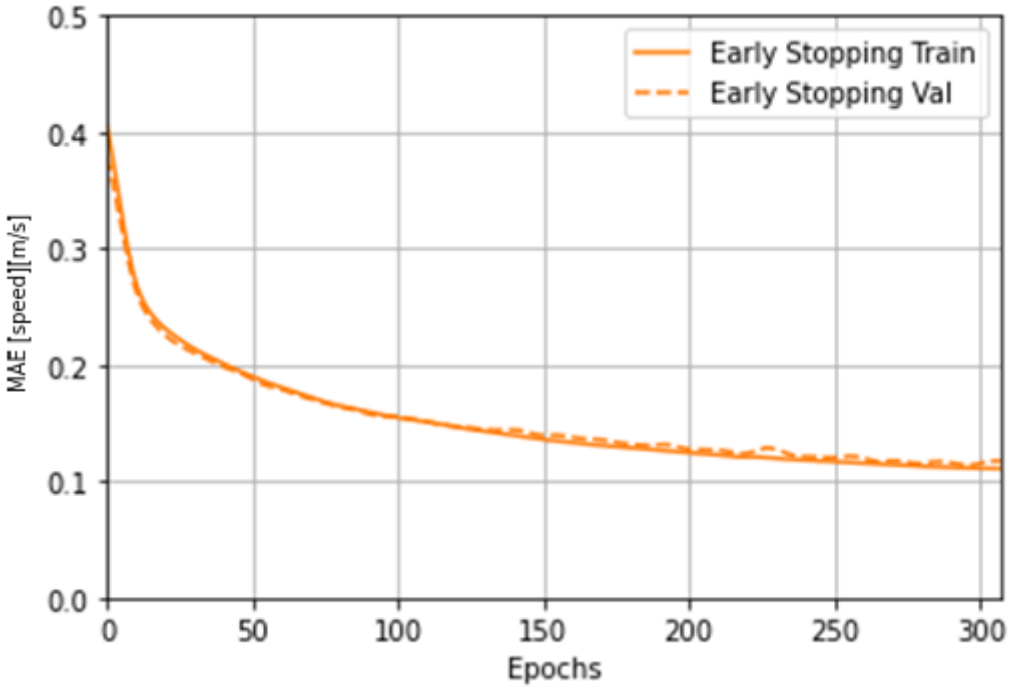

Figure 3.2.2.3: MAE trend over 300 epochs with Grid Search optimization

Let us now analyse the graph of the MAE, which uses the early stopping function and remember that it helps our model to be faster, interrupting the training phase when there is no net improvement set as a plate in the code and we see that compared to the non-optimised model we notice an almost total disappearance of the overfitting and a value of the error on the validation set that is around +/- 0.1 [m/s]. An unsatisfactory result, if compared to the almost ten times smaller value obtained with the Aramco 2.0 dataset, but which remains acceptable if we consider that the Grimech dataset is profoundly different. We note that the early stopping function occurred with a larger number of epochs, 300 epochs, a more accurate but slower learning.

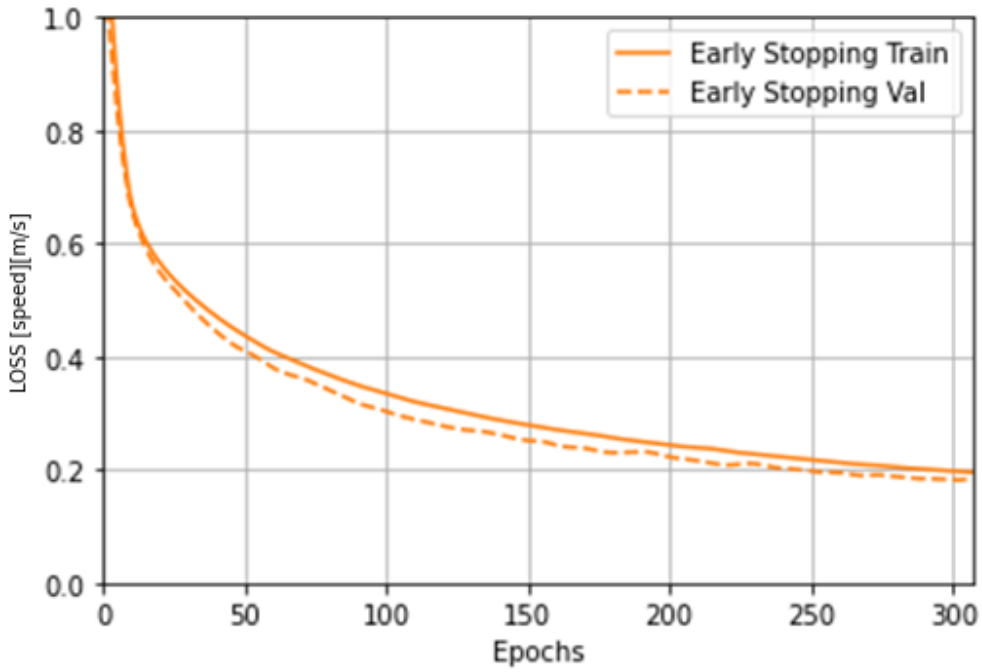

Figure 3.2.2.4: LOSS trend over 300 epochs with Grid Search optimization

Also analysing the loss value with optimized model we see that the number of epochs where our function has stopped learning is around three hundred epochs, which shows an improvement with the training process. The differences, compared to the previous case, we notice when the two curves tend to join, since the beginning of the learning phase sign that our model, tends to make fewer mistakes with the passage of epochs. Both training and validation loss decrease exponentially as the number of epochs increases, suggesting that the model acquires a high degree of accuracy as our epochs increase. Loss value reduced by 30%, with a final value of 0.2 m / s.

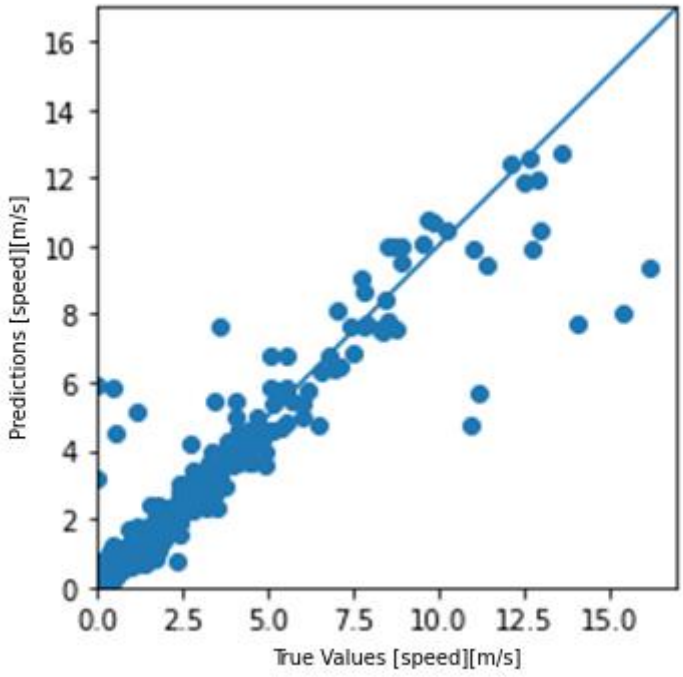

Figure 3.2.2.5: Results obtained by testing our optimised algorithm using Grid Search

Now is the time to evaluate the goodness of our system, using our test dataset, which our algorithm has never seen during learning. Let's see how there is a better fitting in the part with low average speed values and a slight deterioration with higher speed values. From this optimization we can think of a dataset that with certain values of speed, EGR, pressure and dosage do not fully represent most of the data that have contributed strongly to the realization of the connections of our network. If we analyze the starting grimech 3.0 dataset, we notice that there are values with too high speed in the range with pressure of one bar, temperature 1600 K, dosage and EGR 0.4. Values that going to investigate, treats simulations with Aramco 2.0 are not confirmed. This might be one of the main reasons for a non-optimal optimization, but we will keep our dataset unchanged to have an optimal comparison.

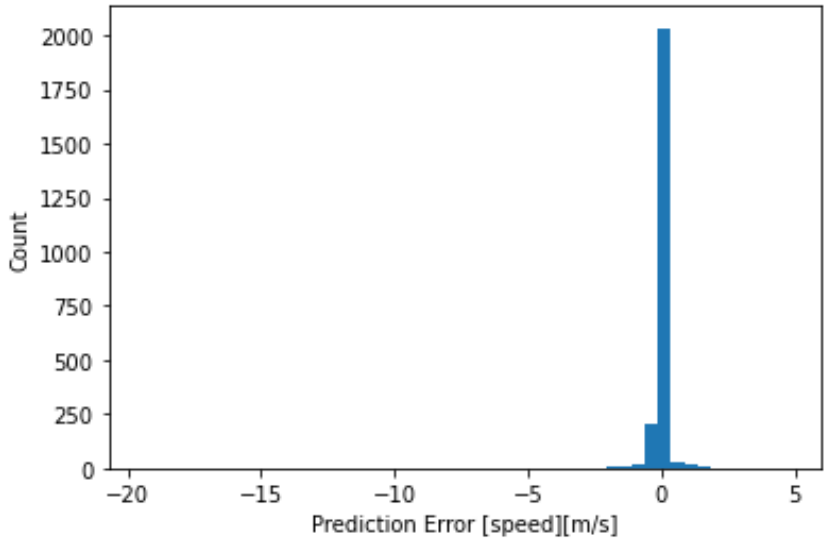

Figure 3.2.2.6: Error distribution obtained by testing our optimised algorithm using Grid Search

We can see that the error distribution also replicates a bar trend and the "prediction" error remains below the range of +/- 4 m/s. A smaller error, if we compare it with the standard result. A result that is an improvement respect the standard model, but not reach the result obtain with Aramco dataset.

3.4 Grid search with Cross Validation

When doing machine learning it is difficult to know in advance which parameters and hyper-parameters will allow a specific model to make better predictions and deciding a priori which model will give better predictions is difficult. In addition, one has to consider for machine learning that the division into different percentages in training and test data affects the quality of the resulting model. These are challenges that need to be considered when optimising our system. There are some tools that make this process easier by making predictions with many different combinations of parameters, hyperparameters, models, training data and test data, returning the combination with the best prediction results, this is very useful to improve our algorithm. After having prepared the dataset, one of the Scikit-Learn techniques, that we can implement in python, more used to evaluate a decision model during this training phase is the so called K-fold cross validation: it randomly divides the training set in K subsets, called folds, and then trains and evaluates the algorithm in K iterations, dedicating K-1 folds for training and 1 fold for validation, always different at each iteration, as we can be seen in the figure below.

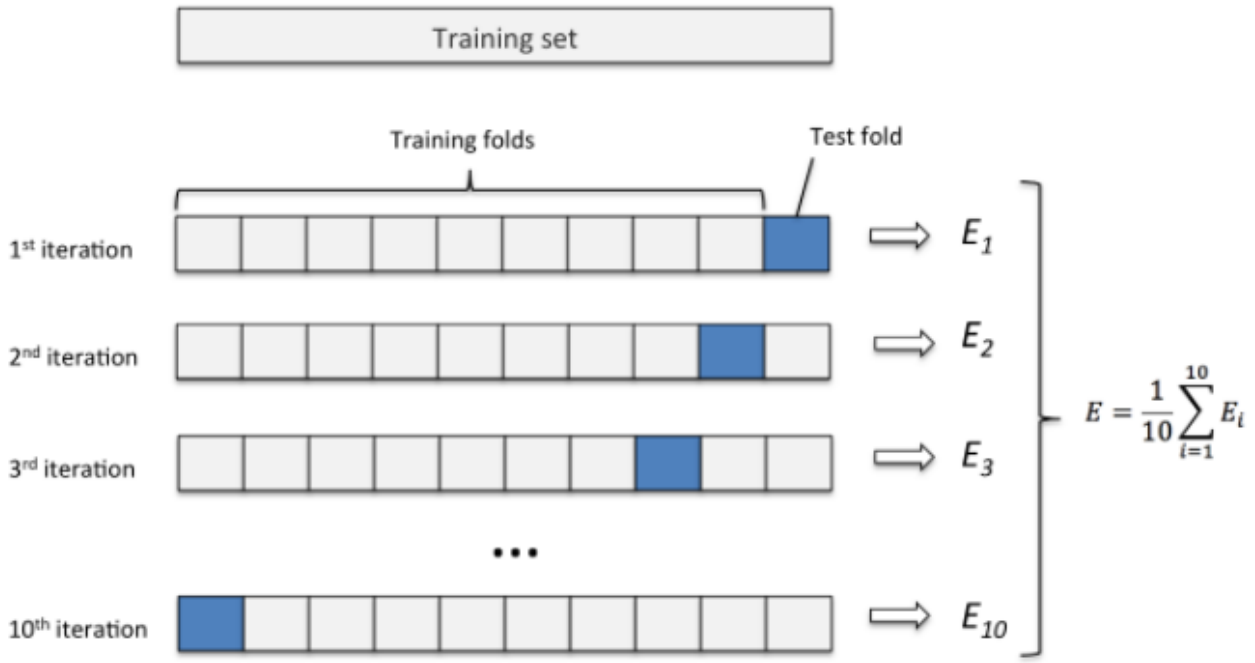

Figure 3.4.1: K-fold cross validation [12]

In our case we have adopted a value of K equal to 8 for the Aramco 2.0 dataset and K equal to 50 for the Gimech 3.0 dataset, which means having the training set divided into eight folds and eight iterations for the first dataset and fifty folds and fifty iterations for the second dataset. The algorithm goes to see which of the iterations used gives the best results (results in score and not in mean absolute error mae or loss). At this point it is necessary to tune the algorithm with the best values of its parameters; this operation is called Tuning. One way would be to manually try different values of the algorithm's hyper-parameters, changing it from time to time until the best combination is found. Obviously, this is a tedious and time-consuming task, since there are functions in Scikit-Learn that perform this task automatically. Such a function is known as GridSearchCV. All we need to do is to assign multiple values to the hyperparameters that will be tested and, consequently, GridSearchCV will find the best combination, using cross-validation, i.e. training on the training set and validating on the validation set. The hyper-parameters used within the code are very important because they allow to increase the predictive power of the model or to build a faster one.

The hyper-parameters used by Scikit-Learn and in the present code are:

- Number of hidden layers
- Number of the neurons per layers
- Optimization
- Learning rate
- Activation function
- Percentage of dropout

Setting the best hyperparameters is necessary to avoid overfitting and underfitting problems, as previously explained in the previous chapters, and is never a simple search and very often requires a high computational time, since the GridSearchCV has to investigate all possible combinations and find the optimal one on a set divided into as many iterations as the K-value of the cross-validation. It is possible to visualise the best combination of the hyper-parameters, also called best estimator, with the function “grid_result.best_score” and “grid_result.best_params” and the results in terms of score and standard deviation. Following the above, our network is trained using the best combination of the hyperparameters found through the GridSearchCV.

It should be noted that thanks to GridSearchCV, we can automate the various iterations, so that we can for example differentiate the dropout levels for each internal level, which is difficult to do in the case of grid search using Hypera. We have to think of advantages in automation and also in code writing but a worsening in the time taken by the model to complete the whole iteration phase. We moved on to designate a scorer object with the function "scoring", in our case creating a function that takes into account parameters used to evaluate regression problems:

- Mean absolute error (MAE) (7)
- Mean square error (MSE) (6)
- R-squared (R^2): is one of the main indices of the goodness of the regression curve and is also known as determination coefficient. R 2 is always less than one and higher than zero. It must compare the training model with the model in which the response is constant and is equal to the mean of the training response

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2} \quad (8)$$

All scorer objects follow the convention that higher return values are better than lower ones. These metrics that measure the distance between the model and the data return a low value, a sign of a reduced distance between actual and target values. It should be pointed out that we use these three parameters, indicated and most commonly used for regression problems, to initially optimise the iterations and for search the ideal hyperparameters for the two different datasets, and then use the mean absolute error (MAE) to obtain our final refit and score.

3.4.1 Grid Search with Cross Validation: results with Aramco 2.0 dataset

The following tables give an account of what the hyperparameters are before and after grid search optimisation.

Hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	64-64
Optimization	RMSProp
Learning rate	0.01
Activation function	Relu
Percentage of dropout	0.2

Table 3.4.1.1: Model standard review

We see how the number of neurons per hidden layer has changed, with a decrease that allows us to resolve the problems encountered with overfitting. In addition, the type of optimiser has also changed, i.e. the way in which we try to reduce the loss between the target value and the real value. We also see that the learning rate and the dropout value between the first level and the second internal level change in value. We can see this result, with the best score, by interpreting the marked overfitting that we had in the non-optimised model, and for this a need to decrease the number of neurons per level and/or decrease the learning rate. We also see that the dropout value in the first level has been increased and decreased in the second level, remembering that increasing the dropout rate forces a shutdown of our neurons that forces a greater learning.

Optimised hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	64-32
Optimization	Adam
Learning rate	0.001
Activation function	Relu
Percentage of dropout 1° hidden layer	0.3
Percentage of dropout 2° hidden layer	0.1

Table 3.4.1.2: Aramco 2.0 model optimised using Cross Validation

We can see below what results have been achieved compared to those analysed in the previous chapter.

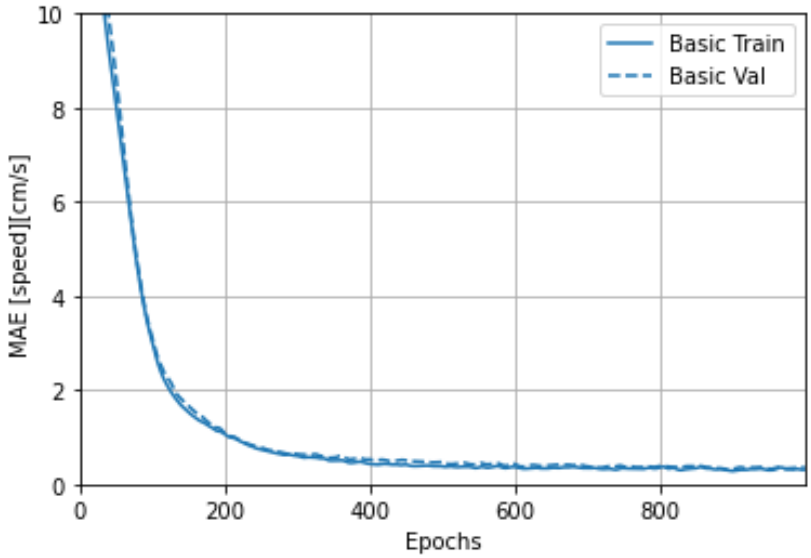

Figure 3.4.1.1: MAE trend over 1000 epochs with Cross Validation optimization

We note that the trend of the MAE along the thousand epochs has completely solved the overfitting problems, and is, compared to the model optimised with Hypera, accurate even after the four hundred epochs. With a low delta between target and actual value. We will see later how the final MAE value changes. We now turn to the analysis of the loss graph shown below.

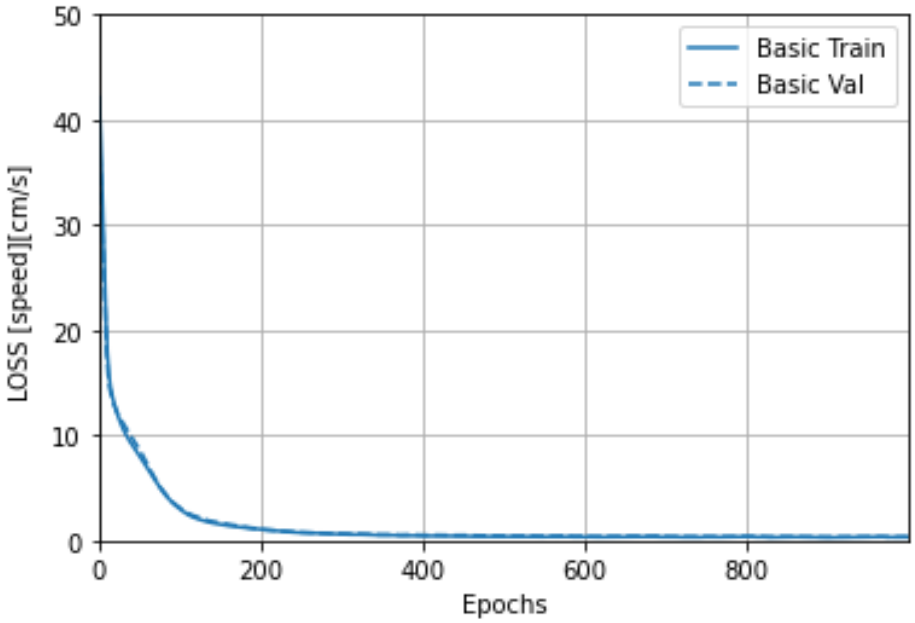

Figure 3.4.1.2: LOSS trend over 1000 epochs with Cross Validation optimization

We note that, compared to the standard model, the overfitting has completely disappeared and that, compared to the model optimised by grid search, it shows a lower loss value from the start of learning. We notice from the almost overlapping of the training and validation curves a better learning during the epochs and a lower final value.

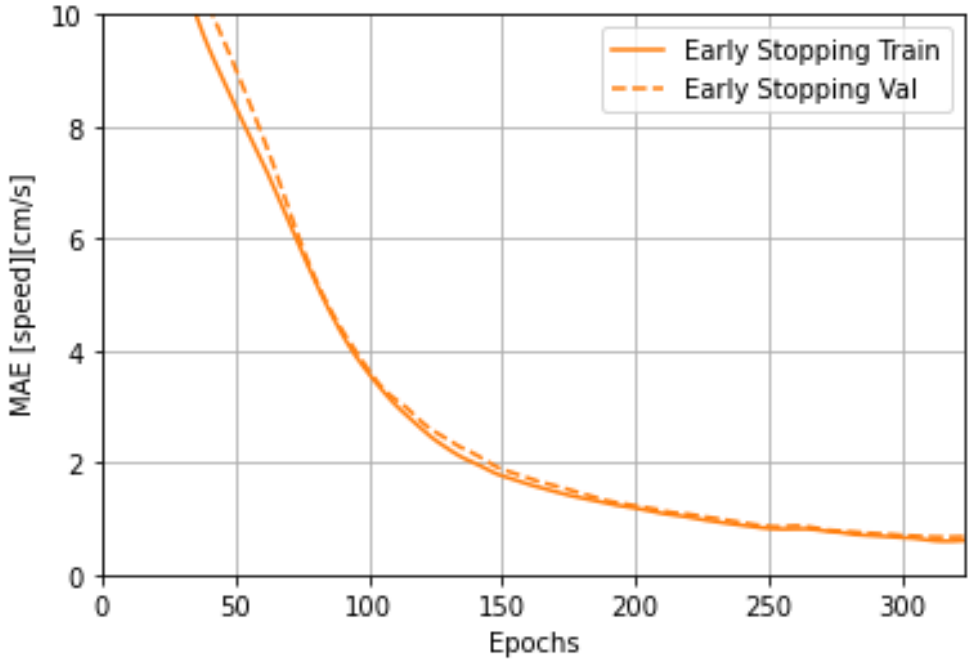

Figure 3.4.1.3: MAE trend over 300 epochs with Cross Validation optimization

Observing the MAE graph, that uses the early stopping function, we can see that the overfitting has completely disappeared, moreover, unlike the model optimised with a simple grid search, we can see that with the passing of the epochs, our model becomes more and more accurate with an ever smaller distance between the target value and the real value. In addition, our final MAE value is smaller than the previously obtained result with a value of about 0.8 cm/s.

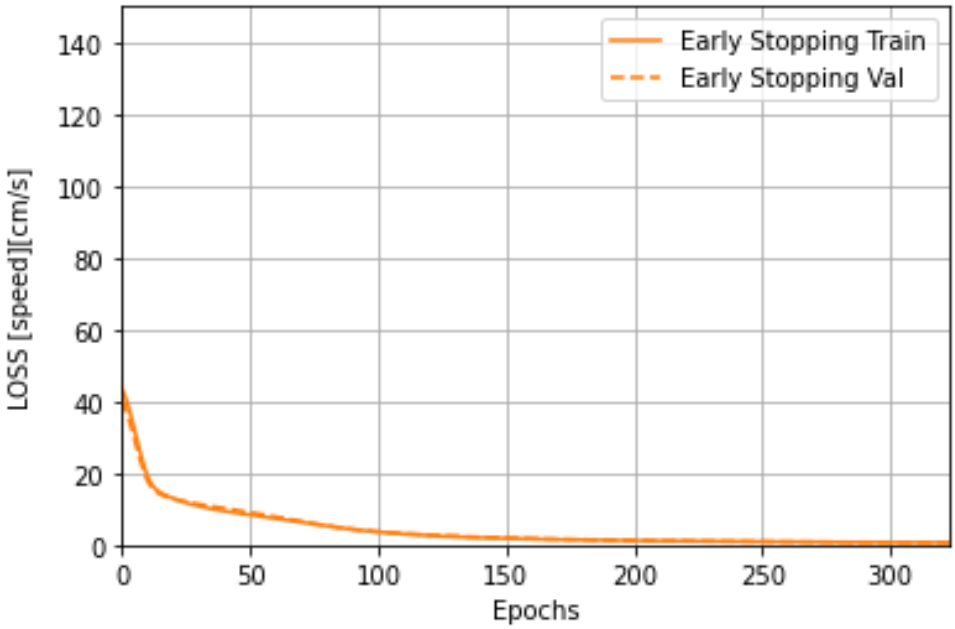

Figure 3.4.1.4: LOSS trend over 300 epochs with Cross Validation optimization

We pass to analyse the graph of the LOSS, we see, as anticipated before, a low loss value since the beginning of the training. We see how the loss value between the objective and real value is very low for the whole range of the three hundred epochs, with the two curves of validation and training superimposed. The thing that stands out the most is a very low final loss value, which is 0.78 cm/s. A truly appreciable value compared to the value of the standard model, with a reduction of almost 90%.

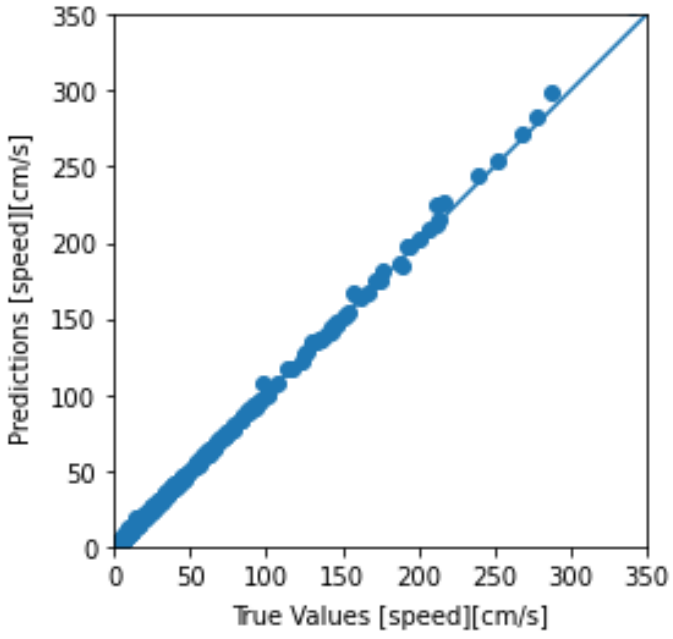

Figure 3.4.1.5: Results obtained by testing our optimised algorithm using Cross Validation

Now it is time to use our test dataset, as we see there is an improvement in prediction for high speed values, confirmed by an almost optimal overlap on our line. We see that there is still a slight worsening in the middle part of the values, it will be clearer later on with the figure showing the distribution of the error of how much the prediction error has decreased in most part of the range.

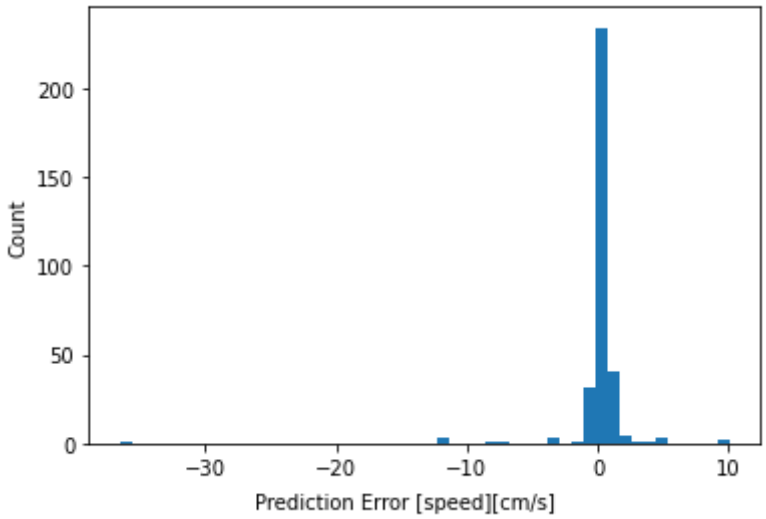

Figure 3.4.2.6: Error distribution obtained by testing our optimised algorithm using Cross Validation

The error distribution is no longer similar to a Gaussian distribution but is much more like a single scatter bar. This means that our model makes a limited error very close to zero, comparing with almost the entire test dataset. There are still a few velocity values where a larger error is evident. However, we are satisfied with the result, with the error distribution curve no longer resembling a Gaussian, indicating a low error value.

3.4.2 Grid Search with Cross Validation: results with Gri-mech 3.0 dataset

The following tables give an account of what the hyperparameters are before and after grid search optimisation.

Hyperparameters	**value**
Number of hidden layers	2
Number of the neurons per layers	64-64
Optimization	RMSProp
Learning rate	0.01
Activation function	Relu
Percentage of dropout	0.2

Table 3.4.2.1: Model standard review

Let's see, from the table below, how the number of neurons per hidden layer has changed, with an increase that as we will see later will allow us to solve the problems encountered with underfitting. The type of optimizer has not changed. The learning rate and dropout value between the first level and the second internal level also change in value. We can see this result, with the best score, interpreting the marked underfitting that we had in the model is not optimized, and thus a need to increase the number of neurons per layer, and/or decrease the rate of learning. We also see that the dropout value in the first level was increased and remained identical in the second level, remembering that increasing the dropout rate forces a shutdown of our neurons that forces more learning.

Optimised hyperparameters	value
Number of hidden layers	2
Number of the neurons per layers	128-64
Optimization	RMSProp
Learning rate	0.01
Activation function	Relu
Percentage of dropout 1° hidden layer	0.5
Percentage of dropout 2° hidden layer	0.2

Table 3.4.2.2: Gri-mech 3.0 model optimised using Cross Validation

We can see below what results have been achieved compared to those analysed in the previous chapter.

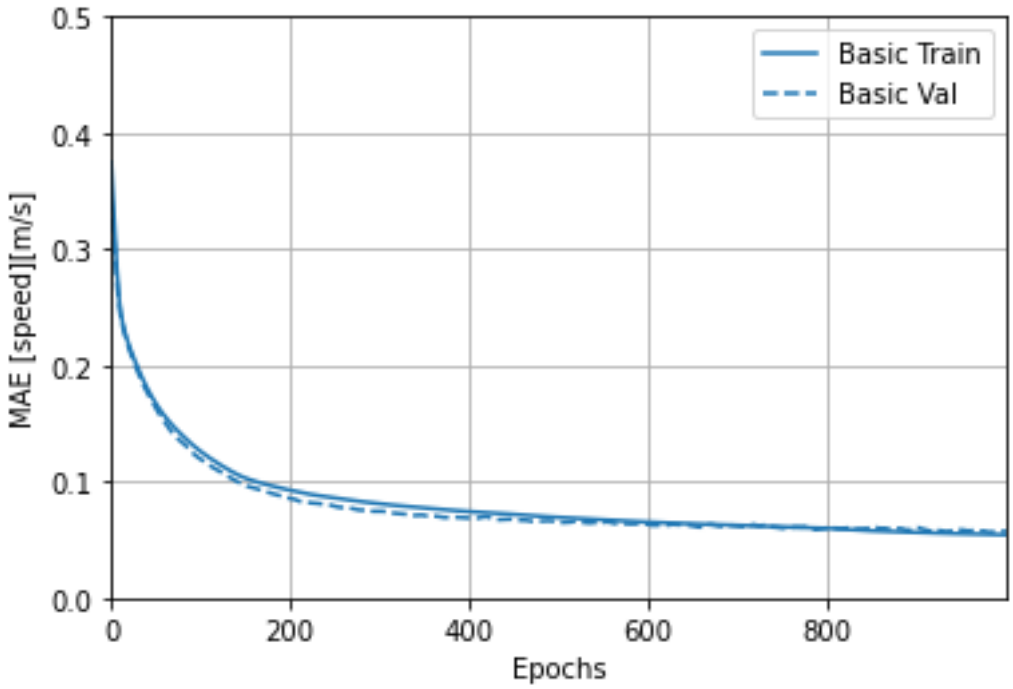

Figure 3.4.2.1: MAE trend over 1000 epochs with Cross Validation optimization

We note that the trend of the MAE along the thousand epochs has completely solved the overfitting and underfitting problems, and is, compared to the model optimised with Hypera, accurate even after the four hundred epochs. With a low delta between target and actual value. We will see later how the final MAE value changes. We now turn to the analysis of the loss graph shown below in figure 3.4.2.2.

We note from the LOSS trend over 1000 epochs that, compared to the standard model, the underfitting has completely disappeared and that, compared to the model optimised by grid search, it shows a lower loss value from the start of learning. The lower underfitting is a great result, if we compare it with the standard model, that has a marked problem with the training process affecting the final prediction result. We notice from the almost overlapping of the training and validation curves a better learning during the epochs and a lower final value. We pay attention also at the starting loss value, that is lower respect the previous case, and confirm the good parameters optimization.

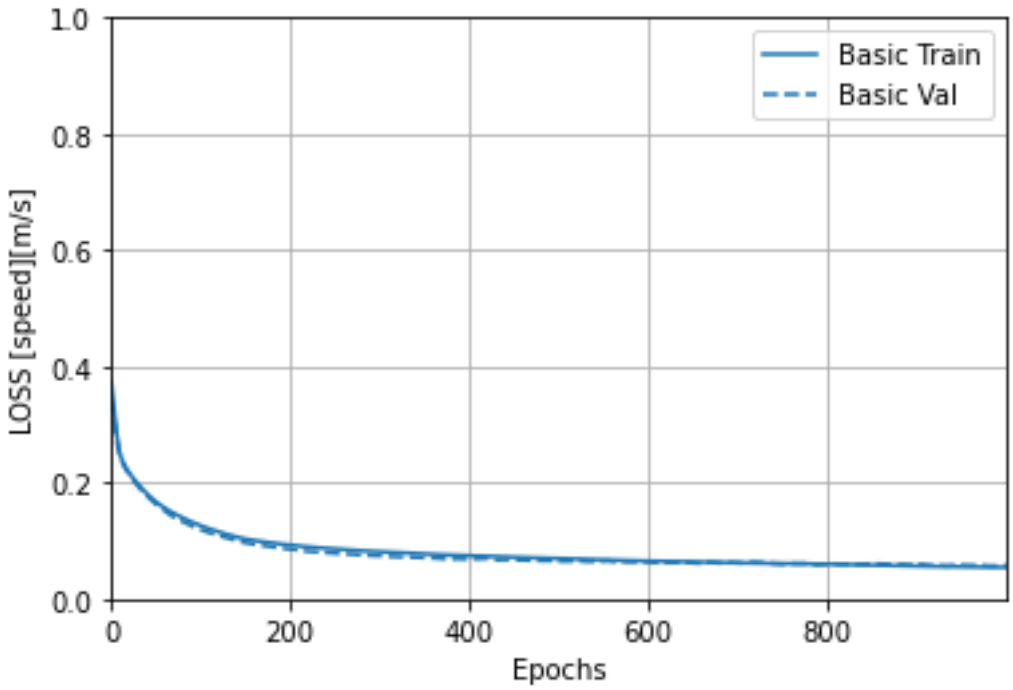

Figure 3.4.2.2: LOSS trend over 1000 epochs with Cross Validation optimization

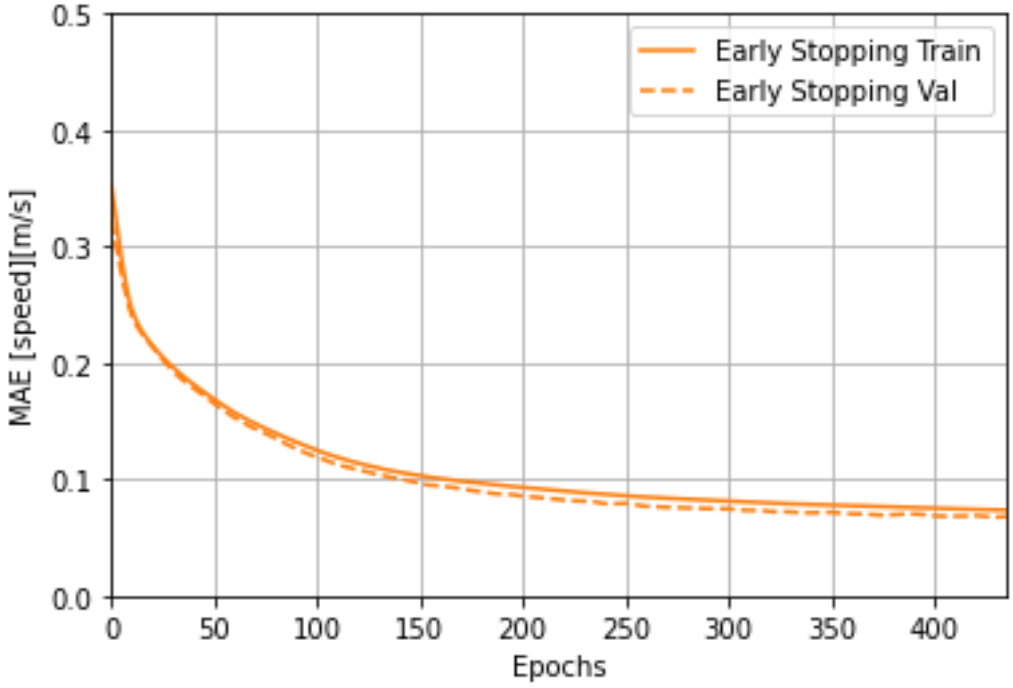

Figure 3.4.2.3: MAE trend over 400 epochs with Cross Validation optimization

Observing the MAE graph, that uses the early stopping function, we can see that the overfitting has completely disappeared, moreover, unlike the model optimised with a simple grid search, we can see that with the passing of the epochs, our model becomes more and more accurate with an ever smaller distance between the target value and the real value. In addition, our final MAE value is smaller than the previously obtained result with a value of about 0.07 m/s, a better result I we compare it with the previous optimization.

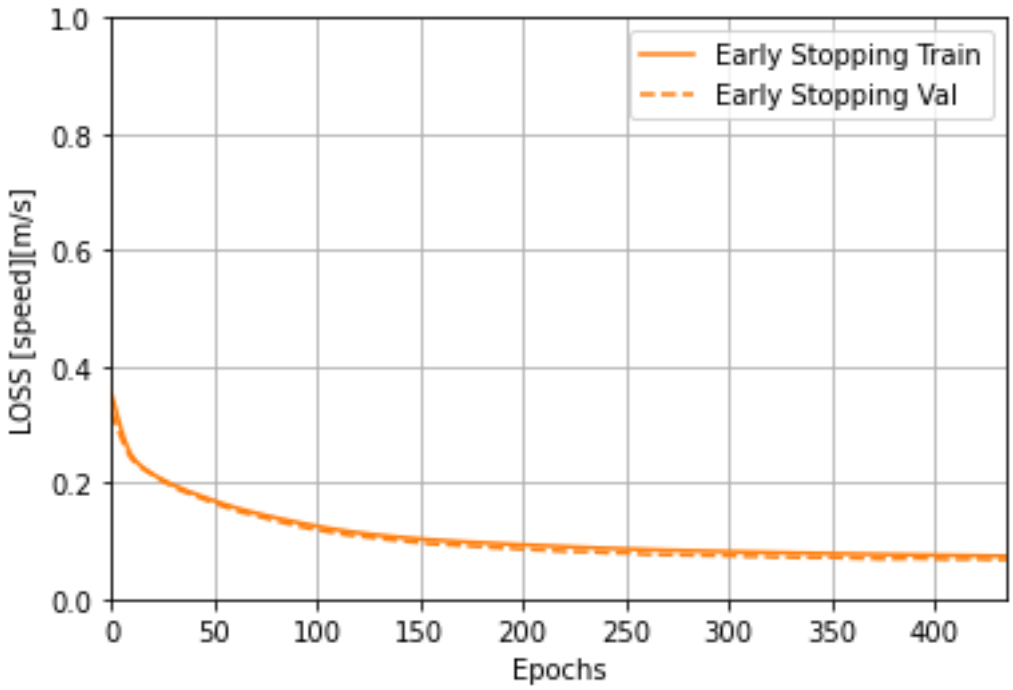

Figure 3.4.2.4: LOSS trend over 400 epochs with Cross Validation optimization

We pass to analyse the graph of the LOSS, we see a low loss value since the beginning of the training, starting from 0.4 m/s. We see how the loss value between the objective and real value is very low for the whole range of the three hundred epochs, with the two the overlapping of validation and training curve. The thing that stands out the most is a very low final loss value, which is 0.065 m/s. A truly appreciable value compared to the value of the standard model, with a reduction of almost 76%.

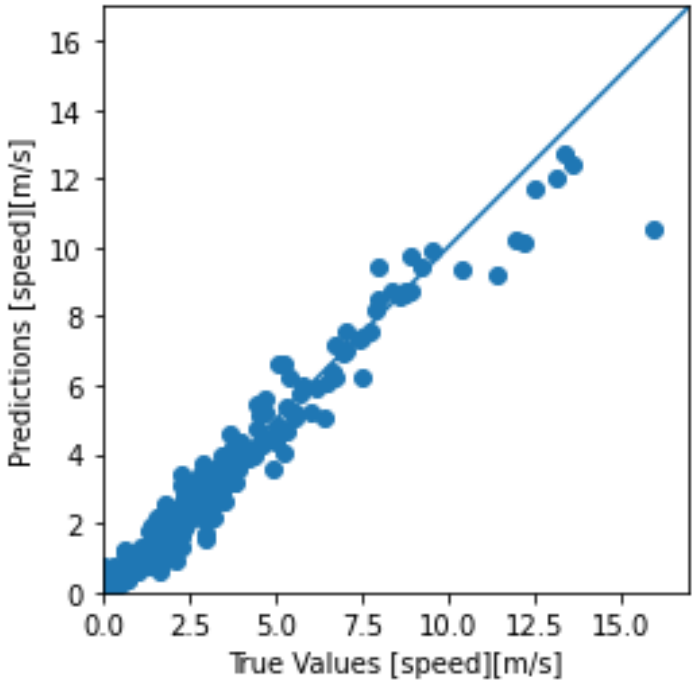

Figure 3.4.2.5: Results obtained by testing our optimised algorithm using Cross Validation

Now it is time to use our test dataset, we can see there is an improvement in the prediction of speed for both low and high values of the latter. We did not achieve a similar and appreciable result as in the case of the

Aramco dataset, but we managed to substantially improve the performance of our model. For high velocity values there is still some point that is probably not representative and does not respect chemical kinetics, so we can make considerations on a group of values, highlighted previously, that afflicts the goodness of our model. It will be clear with the figure below that show the error distribution, and of how much the prediction error has decreased in most part of the range.

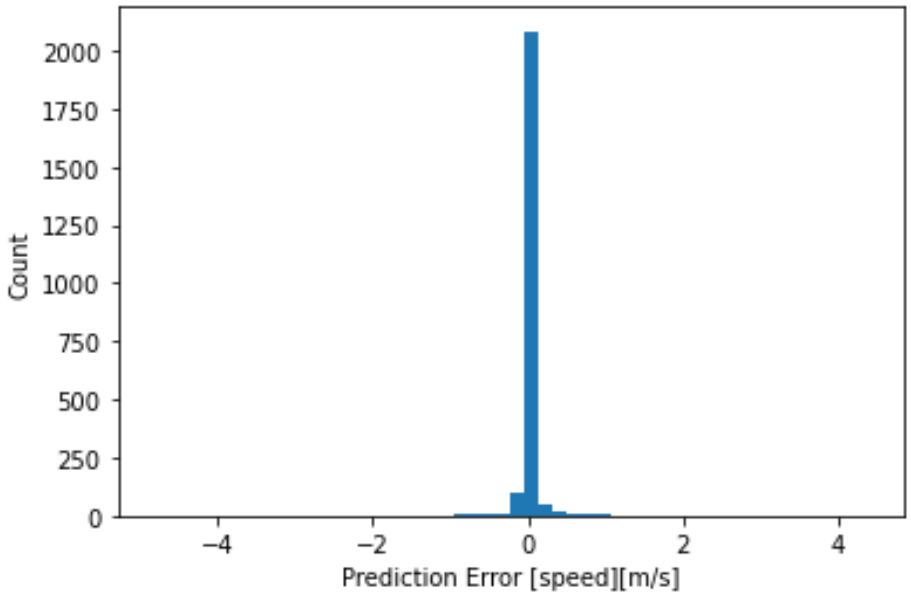

Figure 3.4.2.6: Error distribution obtained by testing our optimised algorithm using Cross Validation

The error distribution is no longer like a Gaussian distribution but is much more like a single scattering rectangle. This means that our model makes a limited error very close to zero, comparing with almost the entire test dataset. There are still some speed values in which a larger error is evident, the result as previously specified of limited values. However, we are satisfied with the result, with the error distribution curve no longer resembling a Gaussian, indicating a low error value.

3.5 Summary of results achieved with our network

In this chapter we are going to report on the performance of our algorithm, considering our three quantities that indicate the goodness of prediction of the model, i.e. the MAE, the LOSS value and the average prediction error.

3.5.1 Summary of results obtained with the Aramco 2.0 dataset

Let's look at the table below to see the progress of our optimisation.

Aramco 2.0 dataset			
Train_test_split = 80%		*Test_size = 20%*	
	Standard model	***Grid search***	***Gr.s. with Cross Validation***
MAE [cm/s]	1.08	1.02	0.80
LOSS [cm/s]	5.11	2.52	0.78
Prediction error [cm/s]	5.31	4.98	3.51
% improvement of MAE		5.55 %	25.92 %
% improvement of LOSS		50.68 %	84.73 %
% improvement of prediction error		6.21 %	33.89 %

Table 3.5.1.1: Results summary obtained with Aramco 2.0 dataset

It must be remembered that these results were obtained with a percentage division of our entire dataset into 80% training data and 20% testing data, and that this was not changed for all the graphs obtained and analysed in the previous chapters. It is worth mentioning that the best fitting, i.e. the best result obtained, for the single folds, during the grid search with cross validation was used to obtain the results concerning the cross validation column. We note a progression and improvement in the reduction of the loss value, i.e. the loss that occurs during training, which reaches a value almost 85% lower than the initial value. This shows us that our model learns well and quickly during the training phase, improving accuracy and minimising time. This last one, the learning time, is very important if we think about the aim of our model, that is to try to find non-linear connections between our input quantities, and to avoid solving and mathematically obtaining this objective. For the loss value, we note that there is an improvement in both grid search and grid search with cross validation. The same cannot be said for the MAE value, which has a net improvement only with cross validation. The latter result may be caused by an improvement in finding the right hyperparameters through a method that splits our starting dataset into many folds and finds the best combination. A not so marked improvement of the MAE should also be analysed in the context of the analysed data, with velocity values expressed in cm/s, and also very high values. A MAE value, i.e. the difference between the predicted value and the actual value, of around one cm/s is already very good. Prediction error also has a tangible improvement with cross validation. In this case, we must emphasise that the results obtained must be interpreted as the distance between the actual and predicted values, so we refer to the distance of our points from the prediction line, as we have seen in the graphs in the previous chapters. The improvement is not clear-cut, even in this case, because in the case of the grid search we had an improvement in the prediction of high speed values and a slight worsening in the average values. With cross-validation we obtained an improvement over the whole range of velocities. An error value of around 3.5 cm/s as a final value satisfies us considering a test size of twenty percent. As you can imagine, by decreasing the percentage of the test dataset our results improve, we will see this later when we vary the "training/testing ratio".

3.5.2 Summary of results obtained with the Gri-mech 3.0 dataset

Let's look at the table below to see the progress of our optimisation.

Gri-mech 3.0 dataset			
Train_test_split = 80%		*Test_size = 20%*	
	Standard model	***Grid search***	***Gr.s. with Cross Validation***
MAE [m/s]	0.12	0.10	0.071
LOSS [m/s]	0.25	0.17	0.065
Prediction error [m/s]	4.10	3.52	1.57
% improvement of MAE		16.66 %	40.83 %
% improvement of LOSS		32 %	74 %
% improvement of prediction error		14.14 %	61.70 %

Table 3.5.2.1: Results summary obtained with Gri-mech 3.0 dataset

We can see that even in this second case, there is a clear improvement with cross validation. It should be pointed out that the Grimech dataset presented some problematic points, with high speed values around a dosage value of 0.4 and high temperature values. In fact, with and without optimisation, we were unable to achieve the predicted speed values, with an error range comparable to the results obtained with the Aramco dataset. As pointed out, these points, as well as the entire Grimech dataset, must be analysed point by point. The objective of using the second dataset is to see how our network adapts to completely different input values. Having recalled these two points, let's move on to an analysis of the table above to see where improvements have been made. We note that the MAE value started from a very high value, which was reduced by about forty percent with cross validation, then analysing our dataset in various subsets called folds, and then changing our hyperparameters. A value that is reduced, reducing the distance between the target value and the real value. The MAE value obtained with the Grimech dataset is still slightly far from the values obtained with the other dataset, where we managed to achieve an accuracy of less than one centimetre per second. If we analyse the loss value, in this case we achieve a reduced final value of about 74 per cent. This value is satisfactory and shows that our optimisation was successful. Again, with a lower loss value, we have a faster and more accurate model in the training phase. The final loss value, as in the case of the mean absolute error, does not reach the values obtained with the Aramco dataset, but we obtain a substantial reduction in the loss value, with all the benefits listed above. When analysing the prediction error, the presence of these trends must be taken into account, as they distort the final results somewhat. In fact, as analysed above, these groups of points have not been eliminated from our dataset in order to have a uniformity of results, from the beginning of the drafting of the algorithm to its optimisation. This choice affects the reading of the values of the prediction error, since it is not possible to see a very clear percentage improvement, but if we analyse the prediction graph, we can see that there is an improvement in the fitting between the real value and the predicted value. This can be seen for both low and high speed values, with a total improvement over the entire speed range. However, there are also points which are far from the prediction value and which affect the final value of the prediction error reported in the table. However, a percentage improvement of around sixty percent is achieved, again obtained by cross validation, which has proved very useful in finding the ideal hyper parameters.

3.6 Sensitivity analysis: k-fold cross validation varying the T/T ratio

One of the key points of this thesis work is to monitor the performance of our algorithm as the size of the training set changes. The assumption behind this procedure is that performance should be better by increasing the size of the training set. Since it is not possible to increase the size and thus the various database instances, as it would also be very time consuming, the performance of the network is monitored as the "train+validation/test split" (t/t ratio) changes. It is clear that by reducing the portion of examples used for testing, more examples will be available for training, and therefore we can understand how an increase in the training portion will improve the performance of the model. It should be remembered that it is essential to use cross validation, we will use the k-folds cross validation procedure, to avoid misleading results caused by a strong dependency on a specific training dataset. In this way we obtain the best result for the k-th fold and use this to choose the ideal hyperparameters and use these to optimise the model.

With less training data, the parameter estimates will have a higher variance, while with less test data, the performance statistics will have a higher variance. In general, we should try to split the data so that neither variance is too high, which has more to do with the absolute number of instances in each category rather than the percentage. Assuming we have 100 instances, we will probably be stuck with cross-validation, as no single split will give us a satisfactory variance in our estimates. If we have 100,000 instances, it doesn't really matter if we choose an 80/20 or 90/10 split, we may choose to use less training data if our method is particularly computationally intensive. Assuming we have enough data to do a proper test (rather than cross-validation), the following is an instructive way to get a handle on variance:

- We split our data into training and testing (80/20 is a really good starting point also suggested by the Pareto Theorem)
- We split our training data into training and validation (again, 80/20 is a fair split)
- We subsample by making random batches of the training data, train the model with these, and record the performance on the validation set
- We try a series of tests with different amounts of training data: randomly sample 20%, say, 10 times and observe performance on the validation data, then do the same with 40%, 60%, 80%. We should notice, both a higher performance with more data, but also a lower variance between the different random samples
- To get an idea of the variance due to the size of the test data, we need to perform the same procedure in reverse. We train on all our training data, then randomly sample a percentage of our validation data a number of times, then observe the performance. Now we should notice that the average performance on small samples of our validation data is about the same as the performance on all validation data, but the variance is much higher with fewer test samples

Remember that in our case we will use the GridSearchCV function in python to obtain the sensitivity results and to analyse how our final MAE and LOSS values vary. Before analysing the table, we must point out two parameters present in it:

- *Best training K-fold score*: this refers to the best result, obtained from the iterations via our GridSearchCV function for the single fold, and is a value between zero and one. The closer it is to one, the more valid and optimal our optimisation is, and we will then use the listed hyperparameters referring to this result to complete the grid search operation in our model
- *Best training K-fold MAE*: this refers to the result of the single fold, and represents our mean absolute error referring to the k-subset.

3.6.1 Sensitivity analysis: different T/T ratio using Aramco 2.0 dataset

Looking at the table, we see that as we thought, there is clearly an interesting trend that seems to confirm the initial theory, i.e. performance increases as the training set increases.

Aramco 2.0 dataset					
		Test size			
		40%	30%	20%	10%
Best Hyperparameters	Number of hidden layers	2	2	2	2
	Number of neurons per layer	64-32	64-32	64-32	64-32
	Optimization	Adam	Adam	Adam	Adam
	Learning rate	0.001	0.001	0.001	0.001
	Activation function	Relu	Relu	Relu	Relu
	% of dropout 1° hidden layer	0.2	0.2	0.3	0.3
	% of dropout 2° hidden layer	0.1	0.1	0.1	0.1
Cross Validation results	Best training K-fold score	0.93	0.92	0.92	0.93
	Best training K-fold MAE [cm/s]	0.13	0.12	0.12	0.12
Final results	Test final MAE [cm/s]	0.89	0.88	0.80	0.78
	Test final LOSS [cm/s]	0.85	0.81	0.78	0.77
	Test final Prediction error [cm/s]	3.89	3.74	3.51	2.95

Table 3.6.1.1: Different T/T ratio with Aramco 2.0 dataset

In our Aramco dataset, we do not notice any particular outliers, which confirms the goodness of the validation of our starting dataset, and confirms that the code written in python optimises our algorithm well. Analysing the best chosen hyper-parameters, we notice that the dropout percentages change when our test size is 30 and 40 percent of our starting dataset. This is a normal consequence of the decrease in data available for training. In fact, if the dropout value were unchanged, we would probably have overfitting as a final result. In addition, we can observe how the scalability of the values is confirmed as the percentage of the test size changes. As a matter of fact, as the test size increases, the performance worsens, with an increase in the final MAE value and in the loss value. We can also notice that the value of the best training k-fold, has a value that oscillates around 0.92, which means that we have a good result in finding the best hyperparameters in our k-fold subset. As a matter of fact, we notice that even with the variation of the percentages of the test size, the k-fold score remains quite uniform, a sign that the optimisation reaches a limit level around a value equal to 0.9, a satisfactory value. Consequently, if we analyse the MAE value of the k-fold score, it too reaches a value of around 0.12, a value relative to our subset that achieves the best results. If we analyse the latter, in relation to the MAE obtained with the whole dataset, we notice that the mean absolute error obtained from the whole dataset is about six times larger than the value of the single fold. This is because the single fold is smaller than the whole dataset, with the size of the single fold being about 6/8 times smaller than the starting dataset. Consequently, the MAE ratios between the single fold value and the whole dataset are also confirmed.

3.6.2 Sensitivity analysis: different T/T ratio using Gri-mech 3.0 dataset

Looking at the table, we see that as we thought, there is clearly an interesting trend that seems to confirm the initial theory, i.e. performance increases as the training set increases, but with some difference.

Gri-mech 3.0 dataset					
		Test size			
		40%	30%	20%	10%
Best Hyperparameters	Number of hidden layers	2	2	2	2
	Number of neurons per layer	128-64	128-64	128-64	128-64
	Optimization	RMSProp	RMSProp	RMSProp	RMSProp
	Learning rate	0.01	0.01	0.01	0.01
	Activation function	Relu	Relu	Relu	Relu
	% of dropout 1° hidden layer	0.5	0.3	0.5	0.5
	% of dropout 2° hidden layer	0.1	0.2	0.2	0.2
Cross Validation results	Best training K-fold score	0.87	0.88	0.87	0.91
	Best training K-fold MAE [m/s]	0.013	0.011	0.010	0.011
Final results	Test final MAE [m/s]	0.079	0.080	0.071	0.063
	Test final LOSS [m/s]	0.074	0.077	0.065	0.061
	Test final Prediction error [m/s]	1.61	1.64	1.57	1.49

Table 3.6.2.1: Different T/T ratio with Gri-mech 3.0 dataset

In our Grimech dataset we note small outliers in the MAE and loss values, which we can attribute to the pattern of points identified earlier that also gives problems with the final best fitting. Moreover, it must be remembered that our Grimech dataset has not been validated on time as in the case of the Aramco dataset. As a matter of fact we tried to simulate those outliers points with a different simulation method, to make a direct comparison, but simulating the Grimech outliers points in Aramco did not give us any results, with the simulation not being able to interpolate those points. For this reason, even making a three-way comparison with the kinematics equations was difficult, because if we could not get results from the simulation, we would have to investigate the origin of the problem, which could be related to the calculation sheets used by the two softwares. We should keep this event in mind, considering that the whole dataset has been kept both for the first attempt results and for the results obtained by the optimisation. Analyzing the best chosen hyper-parameters, we notice that the dropout percentages change in the case where our test size is 30 and 40 percent of our starting dataset, this as happened before, is a normal consequence of the decrease of the data available for training. In fact, if the dropout value were unchanged, we would probably have overfitting as a final result. Furthermore, we can observe that the scalability of the values as the percentage of the test size varies, is not completely confirmed, with higher MAE and dropout values in the case in which we have a percentage of test size equal to 30%. In the case where we have normal scalability, we should note that as the test size increases, there should be a worsening of the performance, with an increase in the final MAE value and the loss value. With the values considered as outliers: one for all, the 70/30 t/t split, which shows a peak in the final MAE value, clearly out of trend. We can then simulate the optimisation several times with this test/training split to see if there are any differences. With several simulations with the same set, we can observe something interesting, that the performance does not change much but, with the very first simulations we see a slight increase in performance and therefore a lower MAE value, while those reported are lower with a loss and MAE value very similar to the one in the table. This is because it is believed to be a side effect of the reduced number of attempts, where even with a reduced hyperspace, a finer search will always be beneficial in the long run.

4 Conclusion and future development

The work that has been carried out tries to provide a predictive model of laminar combustion speed obtained through supervised machine learning. Specifically, a code written in Python is used, which exploits functions (in Tensorflow, Hypera, Sklearn) to train the regression model and try to predict, through non-linear connections, the laminar combustion speed. The input data used, organised as a txt file, were extracted by means of two chemical kinetics simulation mechanisms, namely Aramco 2.0 and Gri-mech 3.0 for different values of pressures, unburnt gas temperature, fuel-air equivalence ratio, EGR rate, methane and hydrogen concentration in the mixture. These data then feed our neural network, and are used to select the validation scheme, train and optimise the regression model. The latter can generate responsive predictions for each new dataset if they are organised with the same number of input arguments given during the training activity processed in the application, in fact this is one of the many advantages of using a machine learning model, namely that of adapting to numerous data and input files. In our case, the model fed from the Aramco files had a minus column, because it simulated a dosage value always equal to one. The main results can be summarised as follows:

- The regression model designed thanks to the machine learning algorithms has not only a mathematical meaning, which presents a good goodness of fit, tested thanks to two completely different datasets, but it has also a physical meaning which presents great potential for development. In fact, it was shown, especially with the Aramco dataset, that the speed forecasts achieved excellent results, with predicted and actual speed values very close to each other, and therefore respected the correlations, even non-linear ones, that linked the independent variables together. In fact, linear regression was chosen to find the weights linking the input variables with the final speed value. It must be remembered that our Aramco dataset has been validated and complies with the equations of chemical kinetics with deviations of less than five percent. On the contrary, our Gri-mech dataset has not yet been validated, and from the data contained in it we have obtained worse prediction values from our model. One of the next developments must be a validation of the latter, replicating all the simulations with the Aramco dataset, and then observing if there are any patterns or points that tend to have results with large differences from the equations of kinetics.
- Two model optimisation methods, namely Grid Search and Cross Validation, were observed and implemented, and the improvements of these optimisations were evaluated and quantified. It was seen that in both models, clear improvements were obtained which resulted in our model predicting speed values better and made our model much faster
- A sensitivity assessment was carried out which showed that a decrease in training data leads to a deterioration in model performance. It was noted that even in this case a complete scalability of the results was not obtained in the case of the Gri-mech dataset, which proves to be less reliable than the results obtained with the Aramco data

It might be interesting to test different types of architectures and different types of regression to build Deep Neural Networks to try to further increase performance. Another interesting development could be to implement other types of optimisation to evaluate the performance and look for further optimisation. In addition, the dataset should be extended in the Aramco case, and validated in the Gri-mech case. A further goal could be to include more simulation data as input to our network, perhaps even from different simulation software. Also the choice of programming language for machine learning is quite important, Python is very flexible and reliable in building the networks, in fact, it is one of the most used programs to the architecture and construction of neural networks, but would be interested to try to compare the results obtained using Python with models written and train with other programming languages. In conclusion, artificial intelligence, and in particular Deep Learning, has been a reliable tool to tackle the tasks related to this project, and this technology will transform the way we think and interact with the world.

Bibliography

[1] Deep Learning, Giansalvo EXIN Cirrincione's slides

[2] A. Gèron, Hands-on Machine Learning with Scikit-Learn & TensorFlow, O'Reilly,2017

[3] https://quantdare.com/

[4] https://missinglink.ai/

[5] Complete guide to using Loge Research and simulation software Aramco 2.0 and Grimech 3.0

[6] Shanker M, Hu MY, Hung MS, *Effect of Data Standardization on Neural Network Training*

[7] Peshawa J. Muhammad Ali, Rezhna H. Faraj; "*Data Normalization and Standardization: A Technical Report*", Machine Learning Technical Reports, 2014

[8] https://www.researchgate.net/

[9] https://datascience.stackexchange.com/

[10] Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets"

[11] Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"

[12] https://scikit-learn.org/

[13] A. Cavaliere, Lezioni di Combustione – Parte II. Napoli.

[14] F. Millo, "Appunti di Propulsori Termici," 2019

[15] R. Amirante, E. Distaso, P. Tamburrano, and R. D. Reitz, "Laminar flame speed correlations for methane, ethane, propane and their mixtures, and natural gas and gasoline for spark-ignition engine simulations," Int J Engine Res, vol. 18, no. 9, pp. 951–970, 2017.

[16] E. Spessa, Design of engine and control system's slide

[17] R.Amirante, E.Distaso, P.Tamburrano, R.D.Reitz. Analytical Correlations for Modeling the Laminar Flame Speed of Natural Gas Surrogate Mixtures