POLITECNICO DI TORINO

Master of Science in Automotive Engineering

Master Thesis

Deep learning solution for the performance analysis of hybrid powertrains

Supervisors

Daniela Anna Misul
Claudio Maino
Alessandro Falai
Alessandro Di Mauro

Candidate

Simone Quaranta

March 2021

Abstract

Over the past years, the engine emission regulation has become of crucial importance and the development of Hybrid Electric Vehicles (HEV) is one of the best solutions in the market. Besides the emissions reduction, this class of vehicles has to satisfy important performances constraints.

For this reason, the aim of this project is to develop a new algorithm capable of classifying the admissibility of hybrid electric vehicle layouts, which means to check if the performance index of each layout is lower than a predefined threshold. This algorithm exploits Deep Neural Network (DNNs), an artificial neural network (ANN) with multiple layers between the input and output layers. There are different types of neural networks, but they always consist of the same elements: neurons, synapses, weights, biases, and functions, used to replicate the human brains' behaviour. This structure enables the algorithm to train and learn by itself.

This admissibility classifier has been inserted into an existing pipeline, creating a new one composed of three different consecutive nets: a feasibility classifier, an admissibility classifier and a regressor of CO2. Each layout passes through this pipeline to verify if it satisfies the constraints of emissions (feasibility) and performance (admissibility) and, then, to predict its CO2 emissions.

For this project, Spyder Anaconda has been used: it is an open-source cross-platform integrated development environment (IDE) for scientific programming in the Python language and libraries as Keras and Tensorflow.

Contents

Index of figures

Index of tables

1 Introduction

1.1 *State-of-the-art of hybrid electric vehicle*

Over the last years, the environmental problem grows more and more, and transport sector is considered one of the most responsible of this. The increasing need to make the road greener than before leads companies and market to search for new types of powertrain. Between those studied, the most developed are for sure the electric ones. Electric vehicles can be classified as follows: Battery Electric Vehicles (BEV), Hybrid Electric Vehicles (HEV) and Full Cell Electric Vehicles (FCEV) [1].

Starting from BEV, they are composed of a battery, that is the only energy source, an inverted that transforms DC current in AC current for the AC electric motor, and, then, a mechanical transmission. This solution can be very convenient for what concern the well-to-wheel emissions (WTW), but only if the production of electricity to recharge the battery is totally renewable. In fact, the emissions produced by the vehicle during its life (Tank-to-Wheel, TTW) are closed to zero. This kind of vehicles can be used only for urban cars since the driving range is small, even with batteries with great dimension.

Another category is FCEV; it is a device able to directly convert chemical energy into electricity without combustion neither moving parts, through the electrochemical combination between hydrogen and oxygen, producing water, electricity and heat. The fundamental difference between the Fuel Cell and the battery is that the battery has the reactants always inside (i.e. the weight of a fully charged battery is the same of a fully discharged one), while the FC use reactants coming from the external (not inside the stack): that means that they continue to run as far as they are fueled with hydrogen and oxygen (from ambient air). This technology has a good potential, but is not yet mature.

The last category presented is HEV, a powertrain composed of an internal combustion engine and one or more electric machines. This technology is the most valuable nowadays because unites the advantages of both BEV and ICE vehicles. In fact, the TTW emissions are lower than a normal ICE due to the presence of the electric machine and, moreover, the driving range is higher than a pure BEV because of the presence of fuel tank as additional power supplier. There is a classification inside HEV following the number of traction systems. Series HEV has only one powertrain with an electric machine as torque actuator. The hybridization is realized at the energy source level with an electric link (realized or directly or through a power converter) connecting one electric source (typically, but not necessarily, a battery pack) to an electric generation system based on an ICE mechanically coupled to an e-machine mainly or solely used as generator.

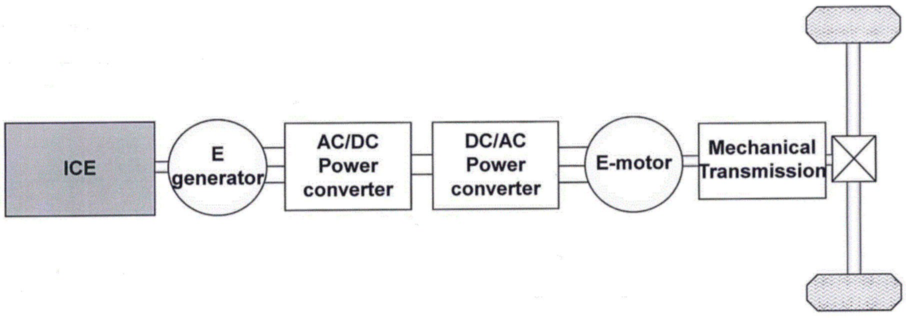


Figure 1.1 - Electric transmission of series HEV - principal scheme

On the other hand, parallel HEV has two traction systems: the one of ICE and the one of e-machine. In particular, depending on the position of e-machine, the parallel architecture is divided in: P1, e-machine connected to the ICE; P2, e-machine positioned between ICE and transmission; P3, e-machine positioned between transmission and differential unit; P4, e-machine position on secondary axle, with ICE one primary one. These architectures are the most widespread in the market and the ones analysed in this project, with special attention for what concern P2 and P4.

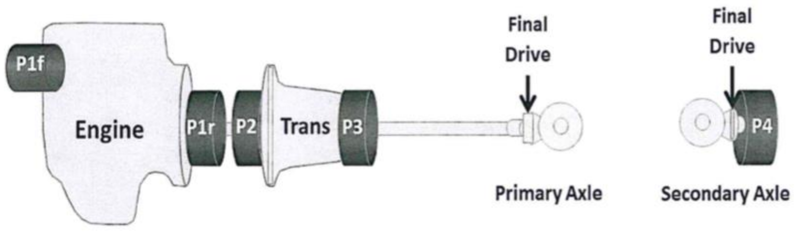


Figure 1.2 - Classification based on e-machine position

It also deserves a quote the HEV complex architectures that unite parallel and series HEV.
The main advantage of parallel architecture is the large amount of different power flows that can be utilize considering the vehicle’s working conditions, as Figure 1.3 shows.

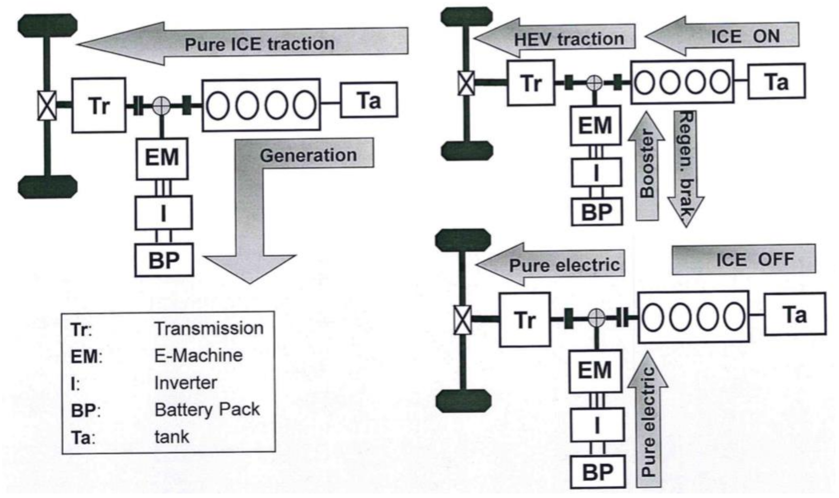


Figure 1.3 - Working modes and power flows of parallel HEV

In order to choose the correct power demand of each component to ensure the best performance, both in efficiency and in emissions, different types of control strategies exist. In literature they are divided in: Global optimization methods, Instantaneous optimization methods and Heuristic methods. This topic has already been developed by the project from which this work is born, so there is no need to repeat the deepen. Anyway, is important to briefly describe the main features of Global optimization methods since it is a starting point also for this project. The aim of this category of methods is to find the optimal strategy by minimizing an objective function over a given vehicle mission, guaranteeing, at the same time, the constraints on the vehicle state variables (e.g. final level of the state of change of the battery). The most widespread technique in this category, especially in hybrid vehicles, is Dynamic Programming, described as a deterministic multi-stage decision-making process, which involves a dynamic system, an objective function and control/state grids [2]. The connection between the techniques described and this project will be clearer in the following chapters.

1.2 Thesis overview

This project was born as a continuation of previous works. The broader idea, as mentioned above, is to create a tool capable of calculating the CO2 emissions of hybrid vehicles, in order to integrate and help a pre-existing tool based on Dynamic Programming techniques. The first network created previously was the regression one, capable of calculating CO2 starting from a dataset composed of only feasible layouts. Subsequently, the need to filter the layouts contained in the datasets, deleting those not capable of completing the considered cycle, led to the design of a new classification network upstream of the previous one, a *feasibility classifier*. Now, a further filtering of the datasets is important, the one related to the performance of the layouts.

Even in a historical moment in which the quantity of emissions is the main parameter monitored for a vehicle, the performance of the engines remains crucial for the achievement of some tasks. For this reason, the study of the performance index of the layouts contained in the datasets, and a consequent classification of it, is fundamental. The purpose of this project is therefore to create a new classifier, an *admissibility classifier,* which will be inserted between the feasibility classifier and the regressor. In this way, the dataset that is given in input to the regressor contains only layouts feasible and admissible.

This project is divided into several sections. The first concerns the description of the datasets used, the second and third explain the theory and method of the tools used. Finally, the results and reflections are presented.

Obviously, some results presented will also concern the first classifier and the regressor, and not only the admissibility classifier, as the study was carried out by also checking the stability of the entire pipeline.

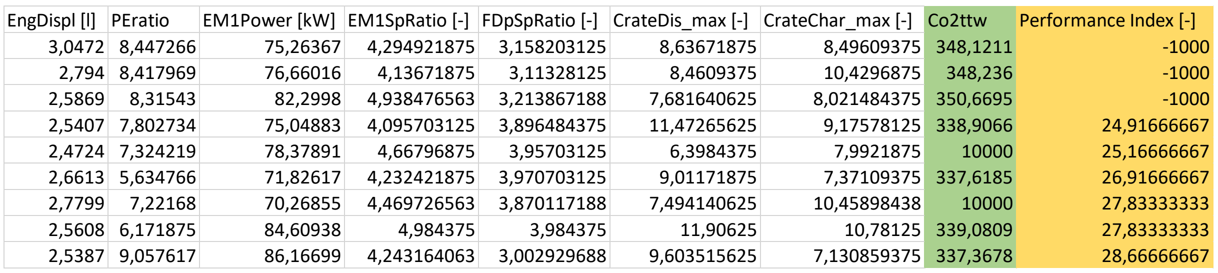
2 Dataset analysis

2.1 Dataset description

As explain in the introduction, the aim of this project is to produce a tool able to simulate what a pre-existing Dynamic programming algorithm predicts, that is CO2 emissions of different architectures of hybrid electric vehicles. This tool works with deep learning neural network based on dynamic programming features. To perform this task, the model has to be trained on data contained in a dataset directly generated by the same DP algorithm.

These datasets are simulated for different hybrid vehicle architectures, those presented previously, P2, P3 and P4. They are divided in two main areas, the first includes the features of the layouts considered, the second the aspects that the tool has to predict, so the CO2 emissions and the performance indexes.

Table 2.1 - Extract of P2 dataset



EngDispl [l]	PEratio	EM1Power [kW]	EM1SpRatio [-]	FDpSpRatio [-]	CrateDis_max [-]	CrateChar_max [-]	Co2ttw	Performance Index [-]
3,0472	8,447266	75,26367	4,294921875	3,158203125	8,63671875	8,49609375	348,1211	-1000
2,794	8,417969	76,66016	4,13671875	3,11328125	8,4609375	10,4296875	348,236	-1000
2,5869	8,31543	82,2998	4,938476563	3,213867188	7,681640625	8,021484375	350,6695	-1000
2,5407	7,802734	75,04883	4,095703125	3,896484375	11,47265625	9,17578125	338,9066	24,91666667
2,4724	7,324219	78,37891	4,66796875	3,95703125	6,3984375	7,9921875	10000	25,16666667
2,6613	5,634766	71,82617	4,232421875	3,970703125	9,01171875	7,37109375	337,6185	26,91666667
2,7799	7,22168	70,26855	4,469726563	3,870117188	7,494140625	10,45898438	10000	27,83333333
2,5608	6,171875	84,60938	4,984375	3,984375	11,90625	10,78125	339,0809	27,83333333
2,5387	9,057617	86,16699	4,243164063	3,002929688	9,603515625	7,130859375	337,3678	28,66666667

Table 2.1 shows a part of the dataset of P2. It is important to notice that some values of CO2 emissions are equal to '10000', meaning that the layout considered is not feasible, so it can't complete the cycle. In the same way, the values of performance index equal to '-1000' mean that the layout is not admissible, so it can't reach the threshold. Considering the Dynamic Programming algorithm from which the dataset is born, a layout is considered admissible if it satisfies some constraints:

- Maintain a pure thermal mode until the end of available fuel, with a maximum speed of 85 km/h with no slope of the road.
- It must sustain for 5 minutes the max speed (85 km/h).
- It must maintain the power split mode for 2 minutes with both a slope of 7.2% and a speed of 60 km/h, and a slope of 14% and a speed of 30 km/h.
- It must maintain the pure thermal mode for 2 minutes with a slope of 14% and a speed of 30 km/h.
- It must take 12.5 seconds to accelerate from 0 km/h to 50 and 28.5 to accelerate from 0 km/h to 80 km/h.

Now it is important to understand the meanings of the features presented in the dataset:

- EngDispl: it represents the engine map to be used.
- PEratio: ratio between electric machine power and the maximum power that can be store in the battery.
- EM1Power: specify the power of electric machine.

- EM1SpRatio: speed ratio between ICE and e-machine.
- FDfSpRatio: speed ratio of the final drive.
- CrateDis_max: maximum discharge of the battery.
- CrateChar_max: maximum charge of the battery.

On the other hand, considering a part of the dataset of P4 (Table 2.2), it is possible to underline the presence of a different feature, FDsSpRatio (speed ratio of the final drive, that is the secondary axle) instead of EM1SpRatio. This is because the architectures P2 and P3 have the electric machine coupled with the ICE that act only on the primary shaft, that is also the final drive, while P4 architecture has the e-machine couple with the ICE only through the road, meaning that there is a mechanical decoupled between them. For this reason, the e-machine acts on the secondary shaft.

Table 2.2 - Extract of P4 dataset

EngDispl [l]	PEratio	EMsPower [kW]	FDpSpRatio [-]	FDsSpRatio [-]	CrateDis_max [-]	CrateChar_max [-]	Co2ttw	Performance Index [-]
2,6158	5,073242	106,0303	4,142578125	14,79882813	8,759765625	9,287109375	377,1168	32,33333333
3,9209	5,063477	113,208	4,576171875	10,47460938	7,998046875	8,033203125	389,4292	41,83333333
2,9027	5,927734	122,5098	4,94140625	11,62890625	10,72265625	11,42578125	388,5565	34,08333333
2,4762	5,498047	112,5488	4,05078125	15,33203125	6,92578125	9,41015625	378,1617	-1000
4,7029	5,200195	123,4619	4,787109375	12,98242188	6,802734375	10,82226563	405,8382	45,25
3,7813	5,15625	117,9688	4,4375	15,8125	10,6875	8,8125	385,3703	44,08333333
3,3826	5,629883	124,0479	3,052734375	10,12304688	9,380859375	9,556640625	368,7308	41,25
2,9611	5,19043	113,6475	3,126953125	13,53320313	8,431640625	7,271484375	368,6318	-1000
2,5016	5,664063	119,7266	3,703125	13,046875	8,859375	6,796875	373,4135	-1000

The two complete datasets are composed of 1500 layouts and are simulated on the same cycle, World Harmonized Vehicle Cycle (WHVC). During the project, a new wider P2 dataset was presented and used to perform some analysis. It is composed of 7500 layouts, a combination of five different datasets performed on different cycles. A deeper explanation of typology of cycles considered is presented in the next chapter.

2.2 Cycles

At the beginning of the project, as previously mentioned, datasets utilized are simulated only on WHVC. Figure 2.1 shows the vehicle speed profile.

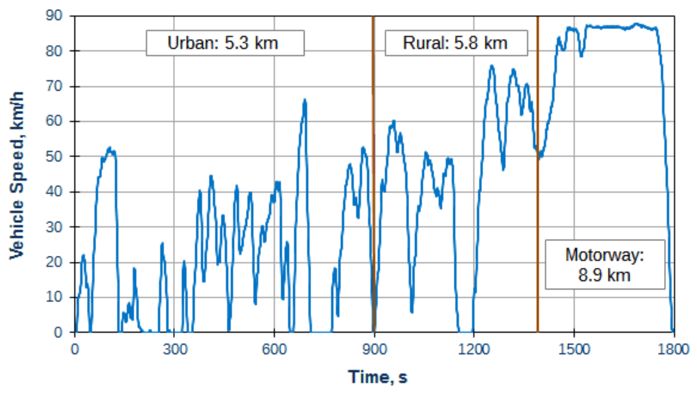


Figure 2.1 - WHVC vehicle speed profile

This cycle is a chassis dynamometer test. It lasts 1800 seconds and is divided into three segments:

- The first 900 seconds represent an urban driving with a low average speed (21.3 km/h) and a maximum speed of 66.2 km/h.
- The second segments, 481 seconds long, represents a rural driving and, for this reason, both average and maximum speed increases.
- The last 419 seconds are defined as highway driving. In this section speeds are high, with an average of 76.7 km/h and a maximum of 87.8 km/h, and starts, stops and idling are absent.

As widely explained during the introduction, it is clear from literature that deep learning neural network increases its performance with the increase of dataset dimension. Obviously, some project can have access to huge group of data, composed of hundreds of thousands or more of them, but this is not the case.

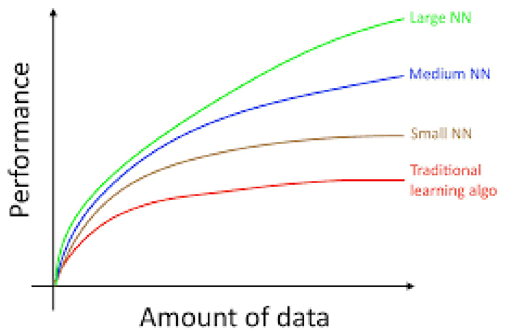


Figure 2.2 - Neural network performance with respect to the amount of data

One of the aims of this project is to increase dimension of the pre-existing dataset in order to improve the performance of the model. Even if the order of magnitude is not comparable with the example mentioned before, a new P2 dataset is been created joining different datasets simulated in different cycles.

A new cycle considered is European Transient Cycle (ETC). It has been introduced for emission certification of heavy-duty diesel engines in Europe starting in the year 2000. As in WHVC, this cycle lasts 1800 seconds and is divided in three segments. The duration of each part is 600s.

- Part one represents city driving with a maximum speed of 50 km/h, frequent starts, stops, and idling.
- Part two is rural driving starting with a steep acceleration segment. The average speed is about 72 km/h
- Part three is motorway driving with average speed of about 88 km/h.

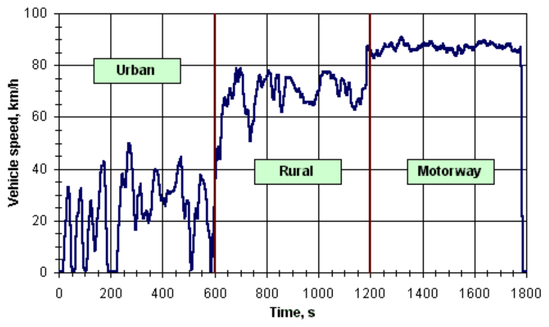


Figure 2.3 - ETC vehicle speed profile

The next cycle presented is Heavy-Duty Urban Dynamometer Driving Schedule (HDUDDS). It is developed for chassis dynamometer testing too. As it can be seen in Figure 2.4 from the vehicle speed profile, its duration is lower than the previous cycles, 1040 seconds.

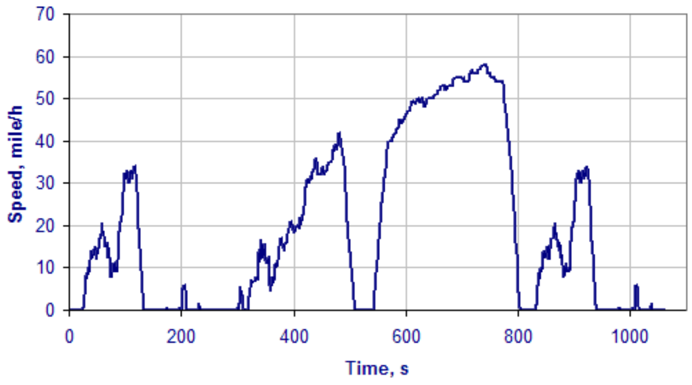


Figure 2.4 - HDUDDS vehicle speed profile

The last two cycles the datasets are simulated on are both chassis dynamometer tests. They are Heavy Heavy-Duty Diesel Truck Schedule (HHDDTS) and City Suburban Heavy Vehicle Cycle & Route (CSC). The first is divided in four speed time modes: idle, creep, transient and cruise (high speed). Figure 2.5 shows them, except for the first.

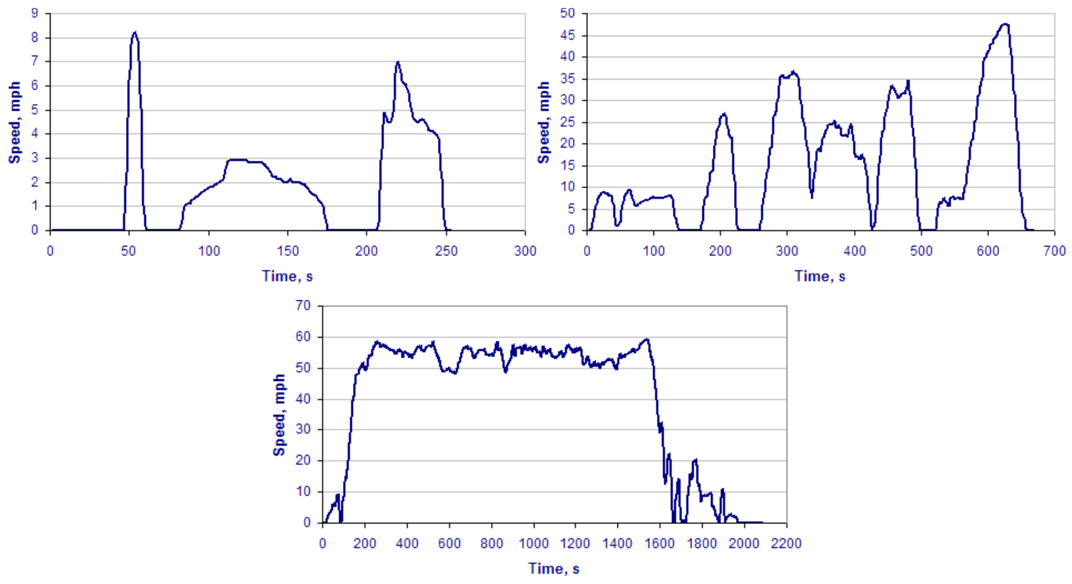


Figure 2.5 - Creep, transient and cruise HHDDTS vehicle speed profile

CSC is also available in route, where the vehicle speed is a function of travelled distance, rather than time.

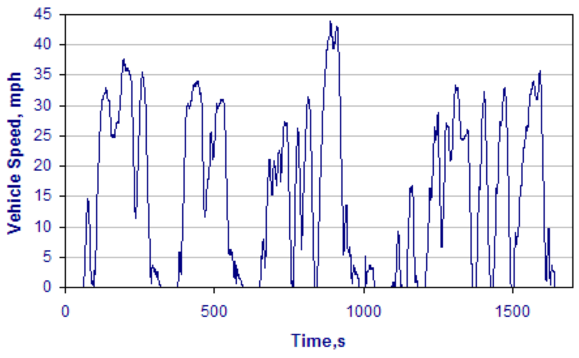
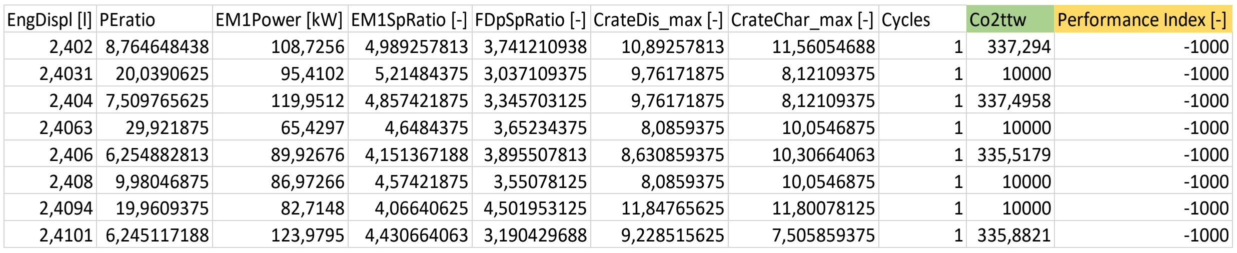


Figure 2.6 - CSC vehicle speed profile

For each of these cycles, a dataset is created and, then, they are joined. The aim of this operation, a part from the increment of the amount of data available, is to add a new feature to the other explained in the previous chapter, a feature that can describe on which cycle the considered layout is performed. An arbitrary value from 1 to 5 is assign to each cycle in the following way:

- WHVC – 1
- ETC – 2
- HDUDDS – 3
- CSC – 4
- HHDDT – 5

Table 2.3 - Extract of new P2 dataset



EngDispl [l]	PEratio	EM1Power [kW]	EM1SpRatio [-]	FDpSpRatio [-]	CrateDis_max [-]	CrateChar_max [-]	Cycles	Co2ttw	Performance Index [-]
2,402	8,764648438	108,7256	4,989257813	3,741210938	10,89257813	11,56054688	1	337,294	-1000
2,4031	20,0390625	95,4102	5,21484375	3,037109375	9,76171875	8,12109375	1	10000	-1000
2,404	7,509765625	119,9512	4,857421875	3,345703125	9,76171875	8,12109375	1	337,4958	-1000
2,4063	29,921875	65,4297	4,6484375	3,65234375	8,0859375	10,0546875	1	10000	-1000
2,406	6,254882813	89,92676	4,151367188	3,895507813	8,630859375	10,30664063	1	335,5179	-1000
2,408	9,98046875	86,97266	4,57421875	3,55078125	8,0859375	10,0546875	1	10000	-1000
2,4094	19,9609375	82,7148	4,06640625	4,501953125	11,84765625	11,80078125	1	10000	-1000
2,4101	6,245117188	123,9795	4,430664063	3,190429688	9,228515625	7,505859375	1	335,8821	-1000

This can lead to an improve of performances of the model, with particular attention to those of the feasibility classifier since there is a connection between features of layouts and cycles.

2.3 Dataset manipulation

After the analysis of the structure of the available datasets and their features, now it is the moment to talk about the procedure that the tool created in this project perform in order to reach the best result in the shortest time. As previously presented, the tool is a deep neural network pipeline composed of two classifiers and a regressor. The dataset is given to the first classifier, the feasibility one, and after a first filtration, passes through the new classifier of admissibility and the regressor. The model created has to be trained, validated and, then, tested in order to act as a neural network, i.e. like a human brain, able to learn on its own the pattern present between features. Even if these concepts have been widely presented in the next chapter about the neural network theory, it is important to make a reference here because the dataset is divided in three parts: training set, validation set and test set. The first action made by the algorithm is to create test set and training-validation set. Thanks to the analysis carried out by the project from which this arises, it is possible to know that the best split is 90% for training-validation set (first datasets: 1350 layouts, second dataset: 6750 layouts) and 10% for test set (first datasets:150 layouts, second dataset: 750 layouts). Downstream of this split, the following is between validation and training set. It depends on the number of folds considered in k-fold cross-validation. Leaving aside the description of cross-validation for a moment, which will be addressed later, thanks again to the previous project the author wants to keep the same number of folds, i.e. 8. This second split leads to a training set composed of 7/8 of training-validation set and a validation set of 1/8 of it.

After the creation of these smaller datasets, it is important to perform another manipulation of them, *normalization of data*. This action is fundamental for two main reasons:

- First of all, observing the extracts of the datasets in the previous chapter, it is clear how the features original values are different one with respect to the other. One aim of normalization is to ensure that the magnitude of these values is more or less the same. This leads to a quicker optimization of the parameter of the model and therefore to higher speed of learning. As it is possible to understand from the sequel chapter, gradient descend depend on:

$$\frac{\partial L}{\partial w_i} = x_i * (a - y)$$

  Where x_i is value of the ith features. From this is possible to underline the dependency of the learning speed on the magnitude of features values.

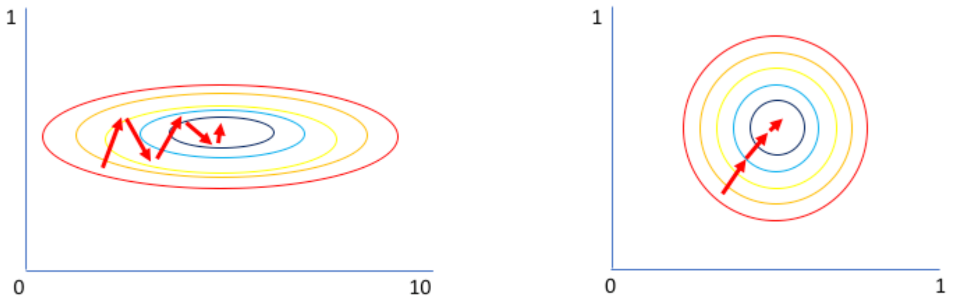


Figure 2.7 - Gradient descent path without (left) and with (right) data normalization

  In Figure 2.7 is represented a very simple model with only two features. In this project case, the number of features is eight, but the visual concept is the same. Thanks to the normalization, each feature gradient has the same emphasis [3].

- The second goal of normalization is to ensure that the values are approximately in the range of "-1" and "+1" to makes learning more flexible having both positive and negative values ready for the next layer of the net [4].

This normalization can be performed in two ways. The first is:

$$value_{norm} = \frac{value - \min}{max - min}$$

Where *min* and *max* are the minimum and maximum values, respectively, of the considerer feature; the second is:

$$value_{norm} = \frac{value - \mathrm{mean}}{std}$$

Where *mean* and *std* are the mean and the standard deviation of the considered feature.

In this project, in which the admissible classifier is developed, the second normalization method is performed to ensure a continuity with the feasibility classifier created in the previous project. For what concern the regressor, done before classifier, the normalization applied is of the first type.

3 Deep Neural Network Theory

3.1 State-of-the-art

In recent years, one of the new technologies that are spreading more in companies is deep learning. As mentioned earlier, deep learning is a sub-category of machine learning, part of the broader branch of artificial intelligence. The goal of deep learning models is to get as close as possible to what is the functionality and structure of the human brain by mimicking the functioning of a neuron and creating functions that operate in the same way.

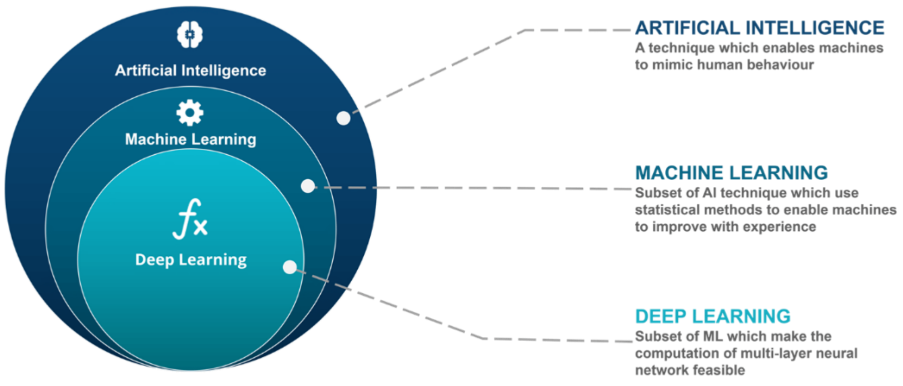

Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 3.1 - Structure of artificial intelligence

The development of deep learning has had several stages since its inception. These deep learning structures, with clear reference to deep neural networks, being the central topic of this project, require very high computation times and this has slowed their diffusion for a long time. The modern development of ever more powerful computers has partially solved this problem, allowing it to spread.

The strong point of these algorithms is their ability to learn the patterns present within the dataset on their own, in order to subsequently make predictions. It goes without saying, therefore, that the more data there are, the larger the datasets are and the more these neural networks are strengthened, increasing their performances.

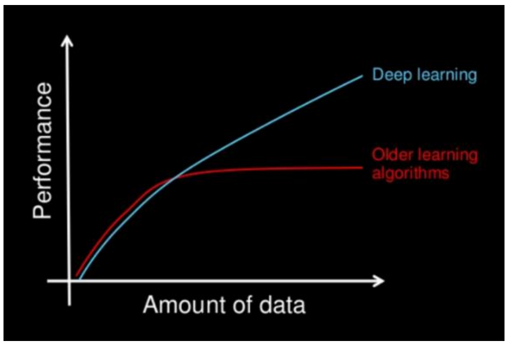


Figure 3.2 - Performance of machine learning models depends on the amount of data

Figure 3.2 proves what one of the leading deep learning experts, Geoffrey Hinton, said:

"Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is"

The first example of this technology is found in 1958 with Perceptron, an algorithm created to classify a group of people into men and women. After making the model learn the main characteristics of the two classes considered, it was able to correctly classify figures that he had never seen. A few years later, Perceptron was questioned, pointing out its limitations, and deep learning research was stopped. Thanks to the writing of the backpropagation algorithm by Jeoffrey Hinton in 1985, he attracted interest in this field again, leading to the creation in 1998 of a convolution neural network by Yan LeCun. From now on, development is rapid. In 2006, Jeoffrey Hinton himself coined the term 'Deep' Learning for the first time to explain the new algorithms that allow the computer to "see" and distinguish objects and text in images and videos. In 2012 another step forward is made by the AlexNet architecture, which wins the ImageNet contest, with an error of about 10% less than the runner-up. Since this transition, these technologies are improving exponentially, also supported by the ever-increasing power of modern computers [5].

A deep learning algorithm can be of two types: *supervised* and *unsupervised learning*. In the first case, a dataset is provided to the algorithm, which, having the results available, learns the patterns present and then replicates them at the end of the training. Two categories belong to this class of deep learning: *classification*, where the algorithm provides as output a discrete value capable of categorizing, recognizing and distinguishing a set of data, and *regression*, in which the algorithm provides a value of continuous output. It is these two types of deep learning that characterize this project.

Also briefly describing *unsupervised learning*, in this case a dataset of results is not provided to the model, which must therefore discover the relationships and patterns that exist between the data. *Clustering* and *anomaly detection* belong to this category.

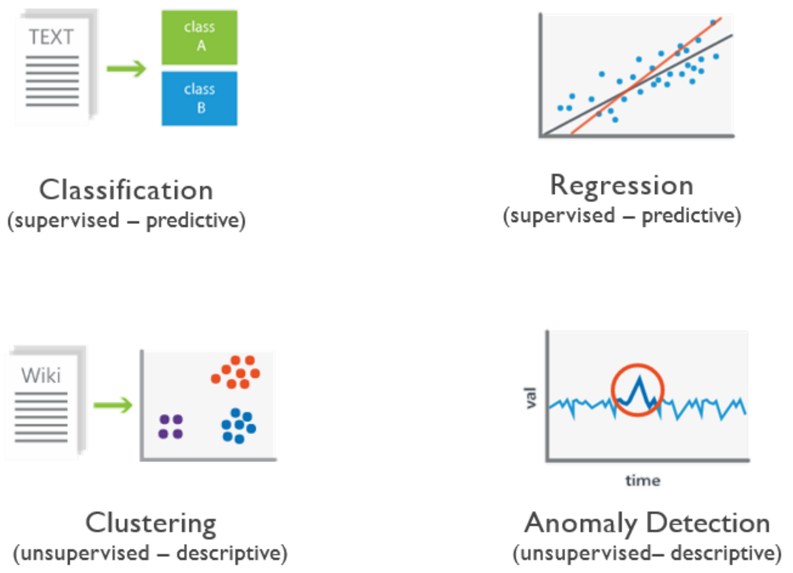

TEXT
class A
class B
Classification
(supervised – predictive)
Regression
(supervised – predictive)
Wiki
Clustering
(unsupervised – descriptive)
val
time
Anomaly Detection
(unsupervised– descriptive)

Figure 3.3 - Different types of deep learning

3.2 Model representation

As explained above, the goal of deep learning, and especially of deep neural networks, is to mimic the behaviour of the human brain as much as possible. This is formed by neurons, consisting of three parts: the nucleus, the dendrites, which are used to pick up the input signals, and the axon, the part that stores the output signal of the neuron itself.

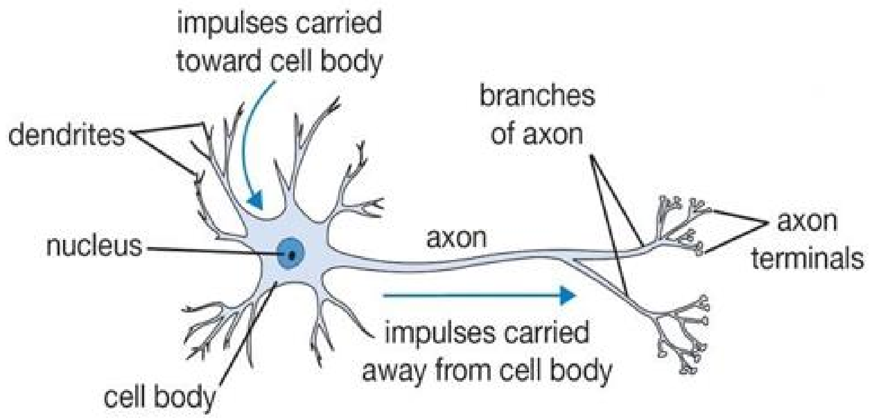

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 3.4 - Brain neuron structure

If now a structure of an artificial neuron is considered, Figure 3.5, similarities are clear. In the place of the dendrites there are nodes, in the place of the nucleus there is what is called the activation unit/function, while in the place of the axon there is an output.

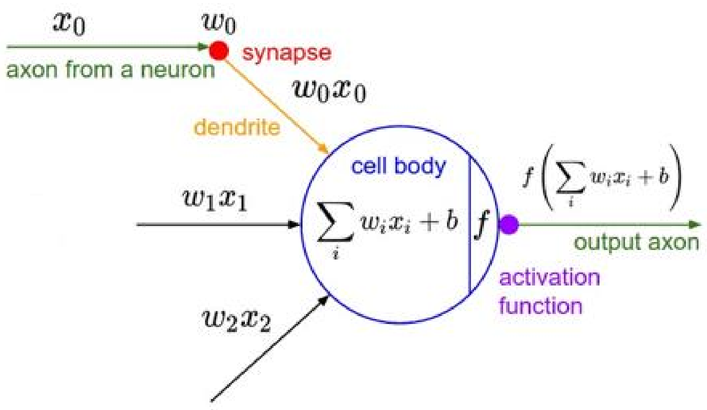


Figure 3.5 - Artificial neuron structure

Where *x* are the input nodes, *w* are the weights of the model, i.e. the parameters that characterize the model. Then there the activation function that gives the output. For what concern x_0, it is called *bias unit* and is always equal to 1.

What represents Figure 3.5 is a simply scheme of one single neuron, but deep neural networks are composed of different layers with a lot of nodes each, Figure 3.6 shows. The first layer is called *input layer*, the last one is the *output layer*. In between these two layers there are one or more *hidden layers,* it depends on the complexity of the model. The number of nodes for each layer is also not fixed. These are two of what are called hyperparameters, which will be explained in more detail in the next sections.

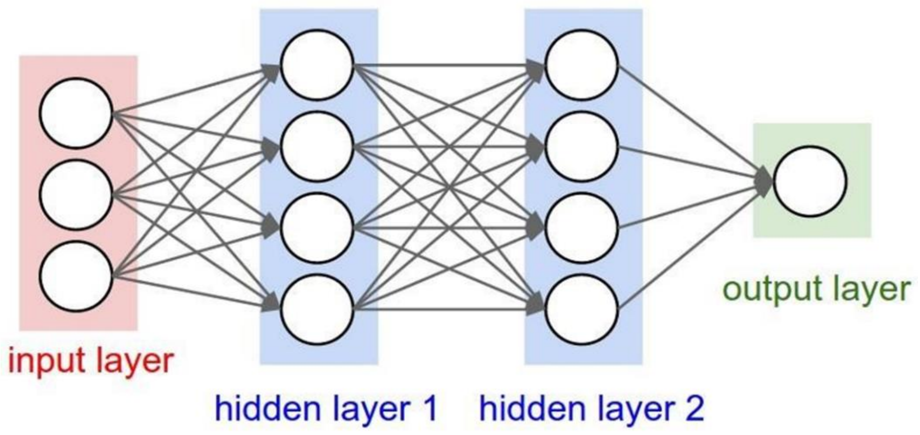


Figure 3.6 - Simplify deep neural network structure

The activation function is used to provide the output for each node, starting from the input data. This function, also called transfer function, maps the output values of the network between -1 and +1 or between 0 and +1 depending on the type of function considered. It is divided into two main categories: *linear* and *non-linear*. In deep learning models, such as the one used in this project, nonlinear functions are used. There are several types of these, but the description of these is not the purpose of this work. Only two types were used in the model definition: the *Sigmoid function* and the *REctified Linear Unit (Relu) function.*

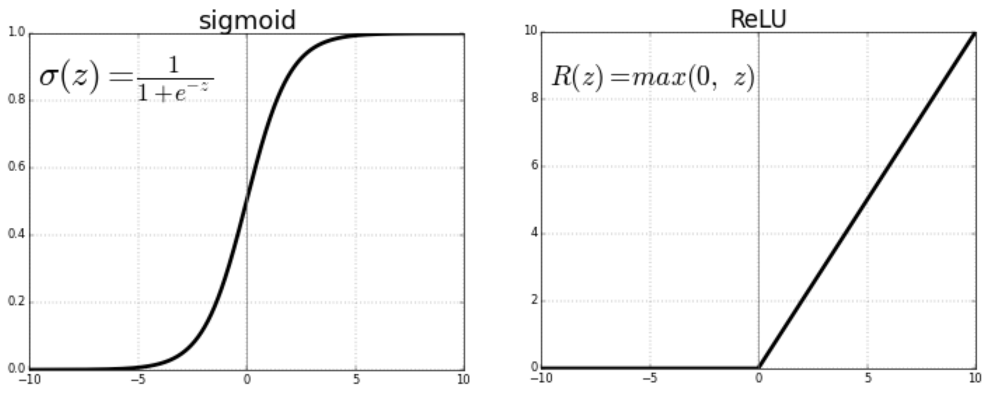


Figure 3.7 - Sigmoid (left) and Relu (right) function

Where:

$$z = x * w$$

is the product between inputs and weights.

The Sigmoid function (Figure 3.7 on the left) exists between 0 and 1. It is particularly used for problem of binary classification, like the one of the *admissible classifier* designed in this project, because values of *z* lower than 0 are classified as 0 and values greater than 0 as 1. However, this outputs technique can lead neural network to stuck.

For this reason, Relu function is the most used. F(z) is 0 for z lower than 0 and is equal to z for z higher than 0. The advantage of this function compared with Sigmoid is that is more computationally efficient because has to just pick up max(0,z) and not does all the calculation that Sigmoid does.

In this project, in the *admissibility classifier,* a Relu activation function is used for the input and hidden layers, while a Sigmoid function for the output layer. This procedure is the same used by the feasibility classifier present upstream, in order to ensure continuity in the pipeline [6].

3.3 Forward propagation

The first step to train a model is represented by *forward propagation*. In fact, it is the process responsible of the first calculation of the output. For each node, the input data enter, pass through the activation function, and exit as output, ready to become input for the next layer. This process is divided in two steps: *preactivation step*, that gives an output called *a,* and *activation step,* that gives n output *h* as output.

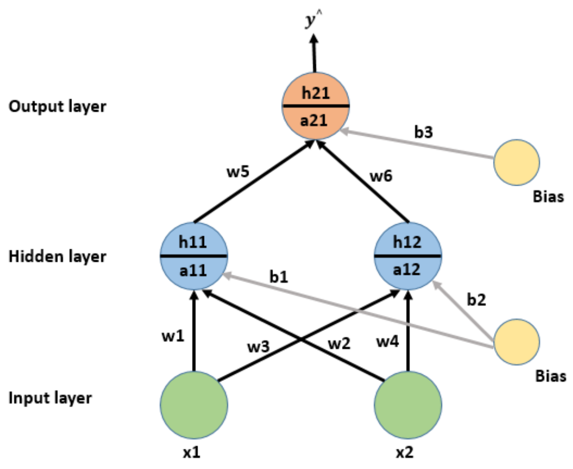


Figure 3.8 - Forward propagation scheme

As it is possible to see in Figure 3.8, the preactivation step is represented by a11 and a12. This step is characterized by the sum of the products between inputs x and weights w:

$$a_{11} = x_1 * w_1 + x_2 * w_2$$

$$a_{12} = x_1 * w_3 + x_2 * w_4$$

After these computations, the values of *a* are passed to the activation functions of the layer considered, in the activation step:

$$h_{11} = g(a_{11})$$

$$h_{12} = g(a_{12})$$

Where *g* represents the activation function. These values of *h* are now ready to become the input values for the next layer, as *x* values for this, until the end of the network structure, when the final output is obtained [7].

Writing the passages above in a more general way:

$$a_j^l = g(\sum_k w_{jk}^l * a_k^{l-1})$$

Where *j* refers to input neuron, *k* to output neuron and *l* is the layer considered.

3.4 *Backward propagation*

The heart of training process is represented by *backward propagation,* or *backpropagation*. Demonstrated for the first time by Jeoffrey Hinton in 1985, its goal is to find a combination of weights and biases to minimize the cost function, as in gradient descent principal.

A cost function J is a measure of difference between what is predicted by the model $\hat{y}$ and the true value y owned in dataset.

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

Where *m* is the number of values considered. This is only a general example of cost function; in the next chapter the ones used in this project are presented.

The direction to take in order to minimize the cost function, that is what kind of parameter combination consider, is given calculating the gradient of it:

$$\frac{\partial}{\partial W_{i,j}^l} J(W)$$

Where *W* represents the parameter combination. This partial derivative measures the sensitivity of J to changes in weights. Considering that the derivative corresponds to the slope of the tangent of cost function, the bigger the derivative, the steeper the tangent, the higher the distance to the minimum of the function.

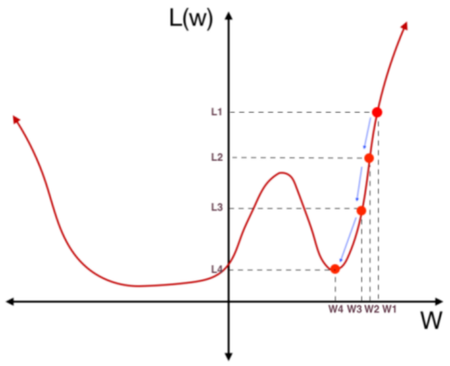


Figure 3.9 - Cost function minima

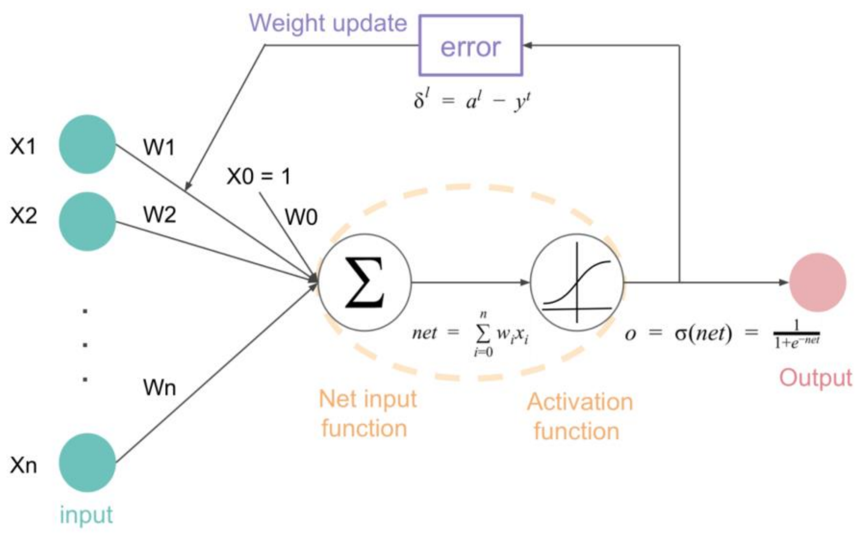


Figure 3.10 - Forward and backward propagation

Thank to Figure 3.10 it is possible to resume what the training process is. It starts with a forward propagation, in which a prediction, and therefore an output, is found. Then the backpropagation starts, in which, calculating gradient of cost function for each layer, a weight updated is performed to minimize the cost function and to reach the best approximation of the model to data utilized.

3.5 Loss function

As mentioned above, the loss function measures the difference between the predicted output value and the true value. This means that it provides a measure of how well the model fits the data, and the smaller the result, the better the model. There are many loss functions that you can use, but describing them all is not the purpose of this project. Designing the admissibility classifier, the loss function used is the Binary Cross Entropy, the same in the feasibility classifier, while the one used in the regressor is Root Mean Squared Error (RMSE).

Binary Cross Entropy is one of the most used loss functions for what concern binary classification. It is calculated as:

$$J = -\frac{1}{N} * \sum_{i=1}^{N} y_i * log\widehat{y_i} + (1 + y_i) * \log{(1 - \widehat{y_i})}$$

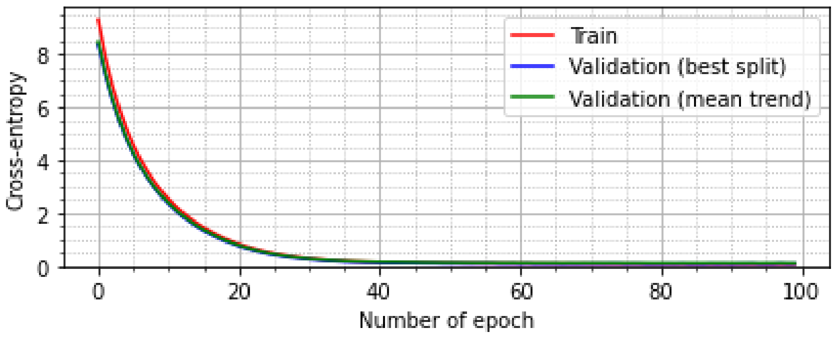


Figure 3.11 - Example of Binary Cross Entropy

Talking about the one of regressor, it is defined as:

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

Where symbols have the meaning explained previously. This function is the most utilized for what concern regression and measures how concentrated the data is around the best fit line [8].

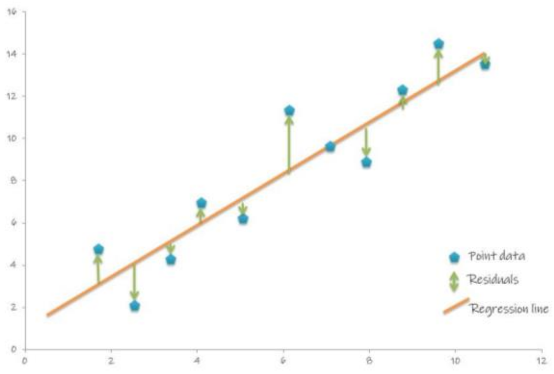


Figure 3.12 - RMSE representation

3.6 Starting parameters and Hyperparameter

In the previous chapters the algorithms used for network training was explained. The main goal was to minimize the cost function in such a way that it fit in the best possible way the data set owned. To do this, the fundamental step is to recalibrate the weights, i.e. to find the best combination of weights, or parameters, that would allow the cost function to reach its minimum. These parameters, or Hyperparameters, are selected within a previously chosen space of hyperparameters through an optimization strategy. Later, in this chapter, we will introduce the hyperparameters used by the network in its training and the optimization strategy. In addition, in addition to the combination of selected hyperparameters, the algorithm uses fixed parameters, set by the author, such as the number of training epochs, the iterations or the number of folds using k-fold cross validation.

3.6.1 Number of epochs

This parameter is usually used as hyperparameter to optimize. In this project is fixed since the beginning. In the last chapter, the results of the pipeline with two different number of epochs have been analysed.

The number of epochs represents how many times learning algorithm passes through the training dataset. The higher this number, the better the model fits data. For this reason, it usually assumes high values, hundreds or thousands; a possible way to set it is monitor when the error is close to zero and, before the increase of it, stop the rise of epochs. In this way it is avoid the problem of overfitting.

Other concepts connected to epoch are the batch size and iterations. When a dataset is very large, it can be a problem to feed a model with all the sample together; for this reason, they are divided in batches and the algorithm passes through them before the update of the other parameters. A batch can be composed of one, more than one or all the samples contained in the dataset. Under these conditions, the learning algorithm is called stochastic gradient descent, mini-batch or batch, respectively.

3.6.2 Learning rate

This hyperparameter is one of the most important. It regularizes the rate of learning of the model helping gradient descent in backpropagation. As it is explained before, during backpropagation an updating of weights is performed in order to reach the minimum of loss function; learning rate gives the importance of this updating. The formula of this process is presented below:

$$w_i' = w_i - \alpha \frac{\delta L}{\delta w_i}$$

Where α is *learning rate* that gives to the algorithm the dimension of the step toward the minimization, w_i' is the updated weight.

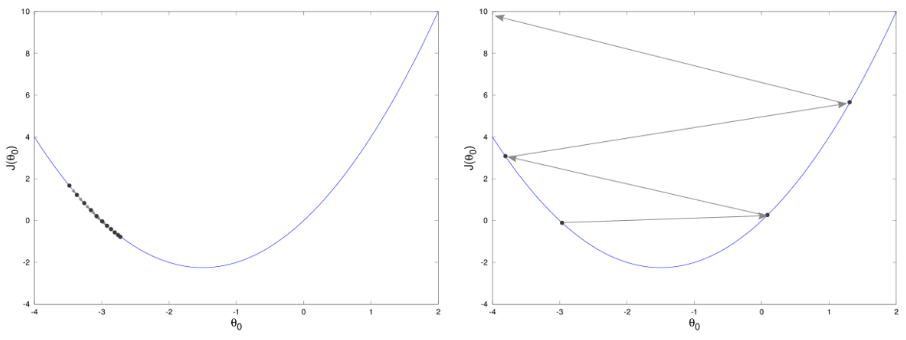


Figure 3.13 - Steps of gradient descent

Figure 3.13 demonstrates two important aspects of the previous equation. Looking at the graph on the right, if the considered point has a negative slope, thanks to '-', w_i' will increase while if it has a positive slope, its value will decrease, converging in any case to the minimum.

The second aspect concerns the size of the steps depending on the learning rate. The smaller the alpha, the more steps the model takes to get to the minimum, risking to run into problems like 'Vanishing Gradient'. Conversely, if the learning rate is too large, it converges quickly but can risk to skip the minimum.

3.6.3 *Number of input nodes and hidden layers*

These hyperparameters are closely related to the architecture of the neural network. Their value depends on the complexity of the model you want to design. If the model is complex, the number of neurons per layer and hidden layers must be increased, however, to the detriment of the calculation speed, which obviously increases [9].

Thanks to the number of the nodes of the input layer, decided by the algorithm as hyperparameter, the quantity of nodes of each layer can be found halving the number of the previous one.

3.6.4 *Dropout*

In this project, a new hyperparameter is inserted in the two classifiers, the *dropout.*

As explained above, the higher the number of hidden layers and neurons, the more the model is able to learn the patterns that are present between the samples. However, if the dataset considered is limited, as happens in this project, an architecture of the model that is too complex can lead to *overfitting* problems.

A model can run into two types of problems, *overfitting* or *underfitting*. The most common is undoubtedly the first. When a model fits the training data too well it is called *overfitting*. This can lead to generalization problems and difficulties in predicting data unknown to the model.

On the other hand, *underfitting* is the opposite problem, that is when the model does not even fit the training data in the right way. This phenomenon is less common and can be solved simply by upgrading the model or by increasing the training time.

The ideal situation, in which the data is modelled in an optimal way, is called *good fit.*

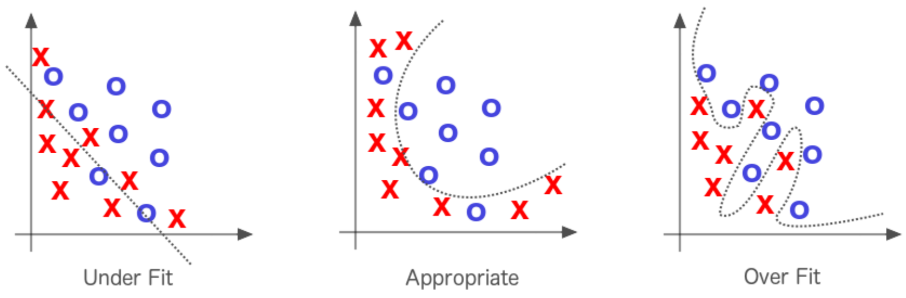


Figure 3.14 - Fitting problem in classification

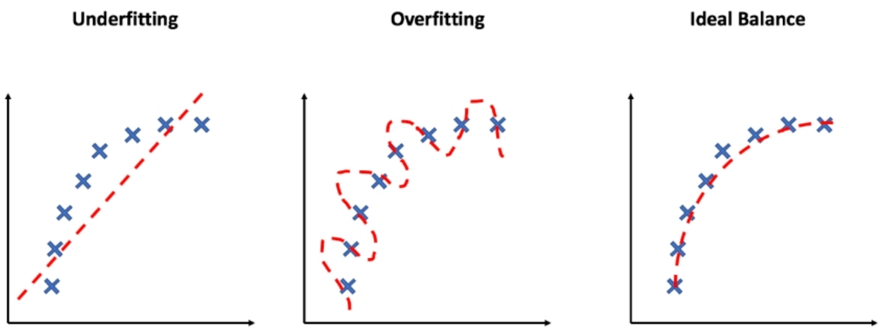


Figure 3.15 - Fittting problem in regression

To solve this issue, *dropout* is a very important technique. It is based on the deactivation of some neurons within the layers during the training phase, and their subsequent reactivations during the testing phase.

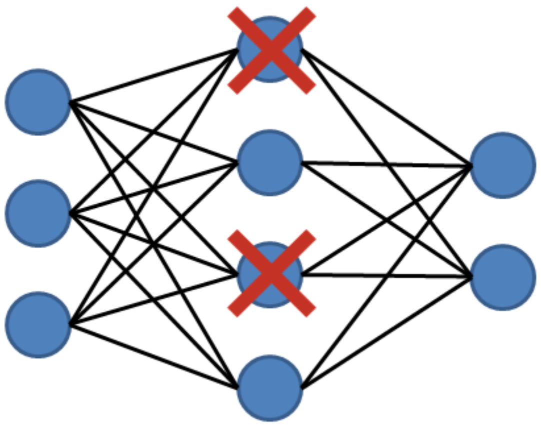


Figure 3.16 - Dropout strategy

This deactivation occurs in a probabilistic way. The value the author applies as a dropout means the probability that each neuron will be deactivated. This operation weakens the model, so as to eliminate rigid patterns linked only to the data used, increasing its ability to generalize other unknown datasets and its robustness of predictions [10]. Obviously, the probability of temporarily deactivating a neuron must not be too high in order not to risk weakening the model too much during the training phase and having strange behaviours in the network, such as a better performance, in the early epochs, of the testing set compared to the training set. A more detailed analysis of this situation has been provided in the chapter on results. A fairly substantial part of the work of this thesis, in fact, is related to the optimization of the right dropout value applied to the admissibility classifier.

3.6.5 L2 regularization

Another importance regularization to solve and prevent overfitting problem is *L2 Regularization*, also considering in the admissibility classifier as an hyperparameter.

This regularization works adding a corrective term to the cost function, that becomes:

$$Cost\ function = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} \omega_i^2$$

Where λ is the regulation factor, the hyperparameter. This technique prevents overfitting because introduces a parameter that is independent of the model and data of the training, increasing the generalize ability of the model [11].

3.7 Search of Hyperparameter

After a brief explanation of the hyperparameters used in this project, it is essential to introduce what is called *hyperparameter tuning*. The type of hyperparameter search affects not only the performance of the model, but also the computational time of the entire algorithm process. There are several research techniques, but their presentation is beyond the scope of this work. In the previous project, the two techniques perhaps most used, the *grid search* and the *random search*, were tested and compared, arriving at the conclusion of the greater effectiveness of the second one. For this reason, the author starts from these results to go on to design the admissibility classification network.

To perform hyperparameters tuning, the definition of a *search space* is needed, where it is possible to select each of them. This space is multi-dimensional, where each hyperparameter defines its own space of values.

As mentioned, the technique used for the admissible classifier designed in this project is random search. It is shown in literature that random search, in addition to having all the advantages of grid search, such as ease of implementation or conceptual simplicity, is much more efficient in the case of large search spaces [12]. In these conditions, in fact, the grid search has a too high implementation time due to all the trials that are carried out with the hyperparameter combinations that form the grid. In random search, the search space is divided in subspaces, and within these, the random combinations of hyperparameters are distributed more evenly, facilitating the search for the optimal one.

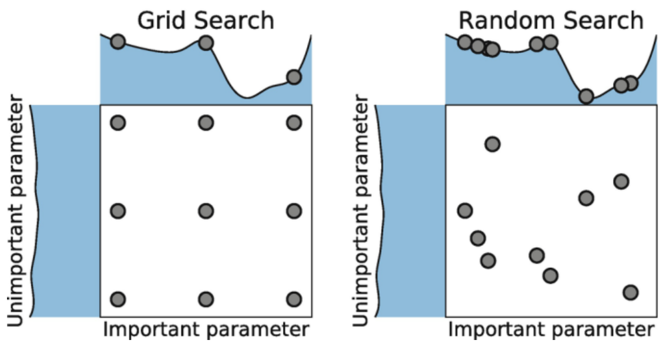


Figure 3.17 - Hyperparameters tuning schemes

In this project, thanks to the research of the previous thesis, the technique is composed of a first random search in the complete search space to find the most promising combinations. Then a new, finer, random search into the search space sectors to find the best between the best.

3.8 K-Fold Cross Validation

Cross-validation is a procedure that, dividing and resampling the training-validation dataset, evaluates the model. The dataset is divided in *k* equal group, or folds, in order to find the best split between training set and validation set.

This procedure works as follow:

- The dataset is sampled randomly and divided in k-folds.
- The model is fitted with a dataset composed of *k-1* folds, the new training dataset, and then validate on the remaining fold.
- The performances of the model with this configuration are computed.
- The following step is to change the composition of datasets, as Figure 3.18 shows, and repeat the procedure computing the other performances. The mechanism is to take as validation set the second fold, then the third until the last one, in order to try all the possible configuration.
- When all the combinations of different datasets are tested, performance results are compared and the best one is taken, obtaining the final division between training set and validation set.

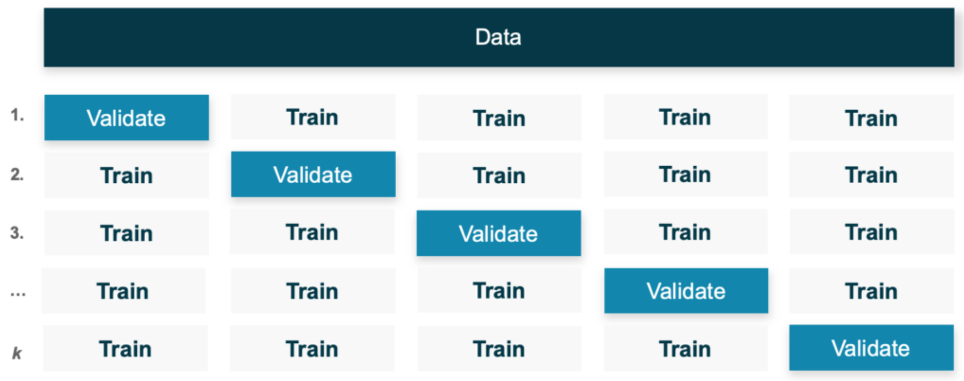

Data
1. Validate Train Train Train Train
2. Train Validate Train Train Train
3. Train Train Validate Train Train
... Train Train Train Validate Train
k Train Train Train Train Validate

Figure 3.18 - k-fold Cross Validation scheme

An important value to decide is the one concerning the number of k folds in which the considered dataset is divided. In fact, a wrong *k* can lead to an high variance of the performance computing at the end of each step, making the measure unreliable and not comparable, or to an high bias, meaning that the model is overestimate. Each fold has to be composed of a good number of samples, high enough to be statistically representative [13].

3.9 Explainable Artificial Intelligence

In recent years, artificial intelligence, and in particular deep learning, have been gaining more and more importance in companies. This continuous development is due to higher performance of these techniques which, however, are accompanied by ever greater complexity. Precisely this topic turns out to be the most delicate. If the decisions made by an algorithm strongly affect people's lives, the reason why it makes certain decisions must be clear and understandable. The emergence of increasingly complex systems, including DNN, has led to an increase in what is called the opacity of algorithms, making them block-boxes. The increase in model performances, which now exceed even those of human, is spreading them more and more. However, this diffusion is hampered precisely by this distrust, on the part of users, of algorithms that they are unable to understand and explain. In fact, interpretability has several advantages, including that of ensuring the impartiality and causality of the choices, as well as the robustness of the model itself.

From this first introduction, it is easy to understand what is the compromise to be met: the increase in the performance of a model corresponds to the increase in its complexity. Consequently, performance and transparency are inversely proportional.

Now it is important to clarify the meaning of some keywords of this topic, so as not to confuse the terminology. The main concept in XAI is that of *understandability*, which also gives rise to *transparency* and *interpretability*. This represents the level a man can reach in understanding a decision made by a model without needs of explaining its structure with explainable techniques. On the other hand, *transparency* represent the feature of a model has to be understandable by itself.

In the literature, XAI is defined as:

> "*Given an audience, an explainable Artificial Intelligence is one that produces details or reasons to make its functioning clear or easy to understand*." [14]

The main reasons that are causing the overbearing development of this technique are essentially two: the first is to apply these models also in the company, and not just in research. They must be explainable and understandable in the best way to ensure that people can trust them. Secondly, understanding the mechanisms behind the reasoning of an artificial intelligence makes possible their improvements in the future.

In the literature a distinction is made between models that can be interpreted already from their design and models that, on the other hand, are explained by means of external techniques. This classification is then made between transparent models and post-hoc explainability:

- **Transparent models** are divided into three levels depending on interpretability degree:
  - *Simulatability*: the ability of a model to be simulated. A model is interpretable if it can be easily presented through text and graphics.
  - *Decomposability*: the ability to explain each part of the model (parameters, inputs, outputs). Each part of the model must be understood by a person without the use of external means.
  - *Algorithmic transparency*: the ability of the model's user to understand the used process to produce each output from the input data. The model must be fully walkable and analysable to enter this level of transparency.

  A model is considered transparent if it is understandable in itself. The models that fall into this category are different, some are presented below:

- *Linear/Logistic regression*
  This model has a direct dependence between real and predicted variables, making the model rigid, and therefore *transparent*. However, to fall within the degree of *decomposability* and *simulatability,* the model must be limited in magnitude.
- *Decision tree*
  Also this model can fit into all levels of *transparency*, depending on its size, having a hierarchical decision-making structure. As the size of the model grows, it moves from *simulatability* to *algorithmic transparency* level.
- *k-nearest neighbour*
  This model deals with classification by predicting the class of a test sample according to the similarity and proximity between the various K examples taken into consideration. This process is similar to that used by people, and for this reason it also falls within the levels of transparency.
- *Ruled-based learning*
  This category includes all the models that generate rules to characterize the input data. Clearly, the rules generated must be *understandable* and *interpretable*. By increasing the number of these and their specificity, the performance of the model increases, but at the same time also its complexity. We are again faced with the compromise presented earlier.
- *Bayesian models*
  This model can be seen as a probabilistic directed acyclic graphical model in which connections represent dependencies between sets of variables. Also this model falls under all levels of transparency only in certain circumstances.

- **Post-hoc explainability:** when the models are not transparent by design, post processing techniques are used that make the models interpretable. These techniques are classified according to the user’s intention, the method used and the type of dataset owned:
  - *Model-agnostic techniques:* these techniques can be applied in a similar way to all models. The main ones are: explanation by simplification, visual explanation techniques and feature relevance explanation.
  - *Post-hoc explainability:* on the contrary these techniques can be applied only to some models: shallow ML models and deep learning models. Referring to these latest models, the continuous development of DNN in search of ever greater performances, has led this type of models to be among the least transparent used. To study them, numerous techniques are used; among these, features relevance methods are among the most used to help the explanation and simplification of networks. This last technique has been presented in more detail in the next chapter and in one related to the results.

It is important to underline the various relevant results that must be taken into account every time an XAI model is created:

- The explanations must be limited (constrictive) meaning that they must explain why one decision is made rather than another.
- Causal connections are more important than probabilistic ones.
- The explanations are *selective*, meaning that it is sufficient to focus only on the main causes of decision [14].

4 Deep Neural Network Model

After a brief description and theoretical introduction of deep neural networks made in the previous chapter, the model of which the design was made is now presented in more detail, with some algorithms and functions used.

4.1 Technical tools

All the pipeline, composed of three different networks, is written using *Python* language.

> *"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics*." [15]

To write neural network using this language, an editing environment is needed. There are a lot of these to be used; in this project *Spyder* is used:

> *"Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features."* [16]

Figure 4.1 - Spyder logo

In python it is important to use what are called *libraries*, packages containing functions already written, very useful to speed up algorithm writing time and to optimize it. These functions are divided into libraries according to their field of use. Those used in writing the pipeline, and in particular the admissibility classifier, that is the topic of this project, are the following:

- ***NumPy:*** it is one of the most utilized packages. This library provides mathematical, logical and other functions using multidimensional array objects, called *ndarray*, focal point of this package.
- ***Pandas:*** another very important and used library is *Pandas*. It is written for data manipulation and analysis. It works with dataframe objects, providing all the functions needed to do operations between them. In this project, considering the continuous manipulation of datasets, it is fundamental and widely used.
- ***Matplotlib:*** considering the need to plot and graphically represent the results of the pipeline, this library is needed. It is a graphical extension of Numpy package.
- ***SciPy:*** this library is used in particular at the beginning of each net, when the search space for hyperparameter is created. In this package is used to solve the most common issue related to scientific computation, like for example a logarithmic distribution between to constraint values.
- ***Scikit-Learn:*** this library contains all the algorithms to process the data, make predictions on them, split dataset in train-test datasets and to compute performance index of networks like *accuracy, cross-entropy or Matthews Correlation Coefficient.*

- ***Tensorflow:*** it is a python low-level library for fast numerical computing. It was created by Google and it is used to create Deep Neural Network models. The version used is compatible only with the version 3 of Spyder, therefore this is the version used, even if the most recent one is the 4th.
- ***Keras:*** this is the last library presented. It is an high-level package that works as a wrapper of low-level libraries, like *tensorflow.*

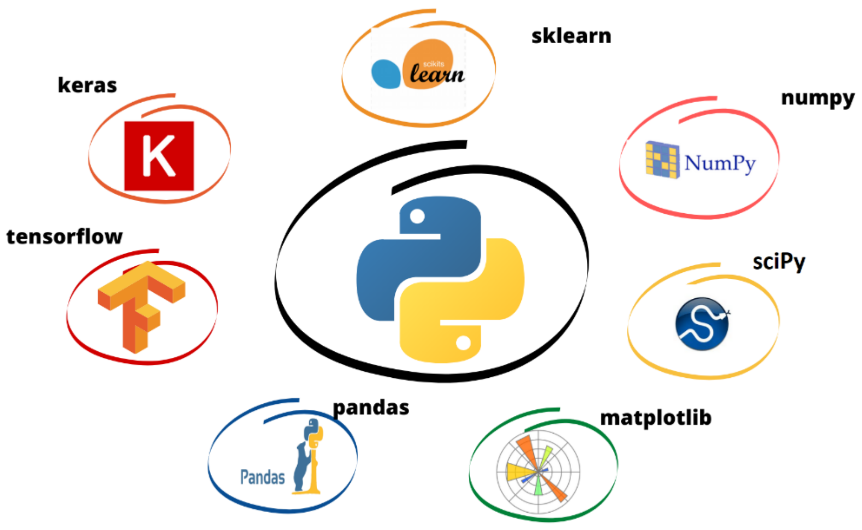
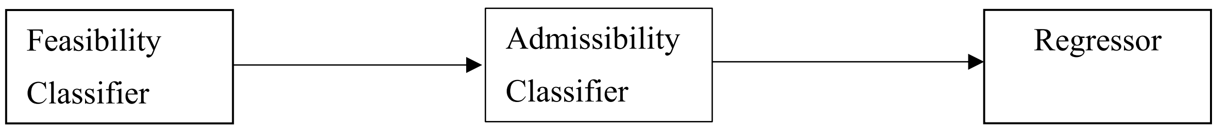


Figure 4.2 - Main Python libraries logo

4.2 Pipeline description

This chapter shows all the main steps that take place throughout the entire pipeline. As previously mentioned, this project starts from a pipeline composed of a feasibility classifier positioned in the first place with the task of filtering the layouts of the dataset to supply them successively to the CO2 regressor positioned downstream of it. This is the starting point of this work, which is developed with the first objective of introducing a new filter between the two pre-existing networks, an admissibility classifier.



The code has been developed so that there is an ON / OFF button on the admissibility classifier, in order to allow the user to decide whether to have a pipeline with or without it every time the program is launched

The first step in the complete pipeline is the acquisition of the datasets by the feasibility classifier. The reworking and manipulation of data is also connected to this action, with the normalization and division of the dataset into training, validation and test set. This favors the training and validation of the model, which is subsequently tested on the testing set. At this point the dataset is prepared for the next step of the second classifier, filtering the 37nfeasible layouts. The new dataset is then composed of:

- Layouts of the training-validation set that have a true value of CO2 different from 10000, the number that shows the unfeasibility of a layout.
- Layouts of test set predicted *feasible* by the model, without what are called *false positive* (they are explained more widely in the next chapters).

Layouts that do not meet these conditions are discarded, while the two layout groups are merged to form the new dataset supplied as input to the admissibility classifier.

Moving on to the next network, the admissibility classifier, the actions performed are repeated in the same way as the previous one. The dataset layouts are again mixed and divided into training, validation and test set. After training, validation and testing of the model, the second filtering is carried out, considering as admissible the layouts that satisfy the following conditions:

- Layouts of the training-validation set that have a true value of *performance index* different from '-1000', the number that shows the not admissibility of a layout.
- Layouts of test set predicted *admissible* by the model, without *false positive*.

Then, the two filtered datasets, the training-validation set and the test set, are passed to the regressor without any further shuffle or division.

Finally, from the training and testing of the regression model, the real value CO2 predictions are calculated.

4.2.1 Performance evaluation

After explaining how the entire pipeline works, it is necessary to present how the results of each network are calculated. In fact, to define the ability of a network to accurately predict data and learn existing patterns in the right way, performance and error indices are calculated. The former must be as high as possible, while the latter, of course, as low as possible.

Furthermore, depending on the type of network, these change. In the classifiers, the *MCC* and *accuracy* performance indices are calculated, while the *cross entropy* is the loss function. The former will be presented later, while cross entropy has already been explained in the previous chapter.

Speaking instead of the regressor, the performance index considered is the *coefficient of Determination R^2*, while the loss function, as already analysed, is the *RMSE*.

R^2 is defined as the difference between a model defined by the simple average of the values of the dataset labels that have to be predicted and the model implemented with the neural network. The higher this difference, the more accurate the model. In mathematical terms, this difference is:

$$R^2 = \frac{SSEM - SSER}{SSEM} = 1 - \frac{SSER}{SSEM}$$

Where SSEM is the *Sum of Squared Errors by Mean line* and SSER is the *Sum of Squared Errors by Regression.*

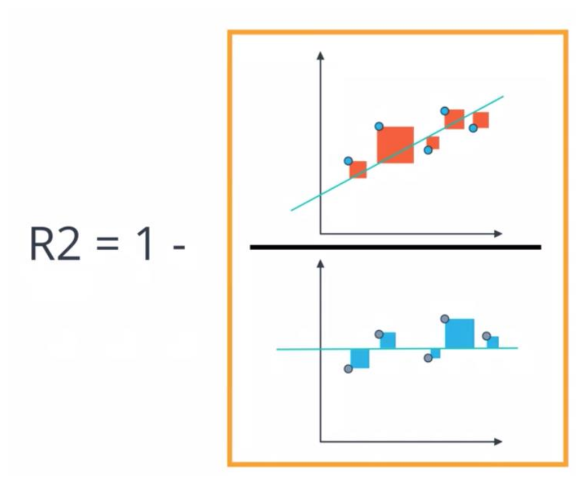


Figure 4.3 - R2 graphical representation

4.3 Hyperparameter search space

As explained previously, the search for the hyperparameters in each of the networks is carried out starting from a multidimensional search space, in which each hyperparameter corresponds to a dimension. In particular, considering six hyperparameter, the search space has 6 dimensions.

In the previous project, the values assumed on average by each hyperparameter were defined after a literature search. Furthermore, this research has shown that some of these have a greater sensitivity to non-linear logarithmic distributions, such as the learning rate or the L2 regularization coefficient. For what concern the possible batch size values, these have been defined with a discrete power distribution of 2. The same principles have been used to initially define the hyperparameter space of the admissibility classifier.

The admissible classifier search space considered for the first analysis is shown below in Table 4.1.

Table 4.1 - Hyperparameter search space of admissibility and feasibility classifier

Hyperparameter	
Hidden layers	1 - 6
Neuron first	30 - 200
Batch size	16 - 512
Learning rate	0.0001 – 0.9
L2 regularization	0.0001 – 0.1
Dropout	0 – 0.3

Dropout is considered as hyperparameter only in a second moment during this project, but is already shown in search space.

Search spaces of feasibility classifier and regressor classifier are considered the same of previous projects. The first one is the same of the one in Table 4.1, while the second is the following:

Table 4.2 - Hyperparameter search space of regressor

Hyperparameter	
Hidden layers	1 - 6
Neuron first	10 - 80
Batch size	8 - 64
Learning rate	0.0005 – 0.5
Weight initialization	Glorot uniform, Glorot normale, Random uniform, Random normal, Truncated normal
Dropout	0 – 0.5

4.4 Optimization tools for model

The model used in this project for the design of the admissibility classifier makes use of several optimization techniques, aimed at preventing future problems. In particular, there is the application of technique to each layer, such as dropout or batch normalization. Furthermore, another technique to prevent overfitting, in addition to that of dropout, is that of inserting what is called Early Stopping.

4.4.1 Dropout

This topic has been extensively explained in the previous chapter and has been analysed again when the results of comparisons between models with different percentage of dropout are presented. However, in this section we want to show where these dropout are applied within the model.

In the previous project, dropout was present only in the regressor, as it was not necessary to add it in the classifier as it was not subject to overfitting. With the introduction of a new classifier in the pipeline, however, the trend of model overfitting has grown, necessitating the introduction of dropouts. To ensure a continuity of technique between networks, the dropout is applied to all the layers, both input and hidden, with the exception of the output one.

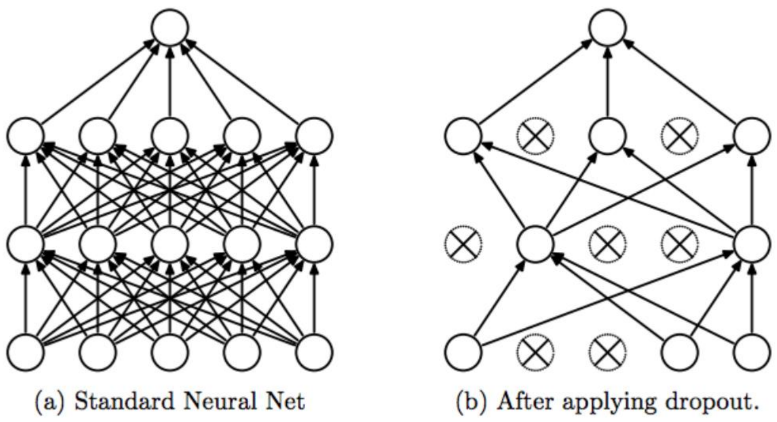


Figure 4.4 - Dropout technique application to all layers

4.4.2 Early Stopping

When the topic of the number of epochs is presented, a possible problem analysed has been the one relating to the choice of the right number of them, to avoid overfitting (too many training epochs) or underfit (too few training epochs). This problem can be solved introducing *early stopping.*

This function monitors a performance indicator, chosen by the author, to check if it improves. When it stops, the early stopping stops the training, avoiding the overfitting and stopping the model in the epoch in which the best performance is reached.

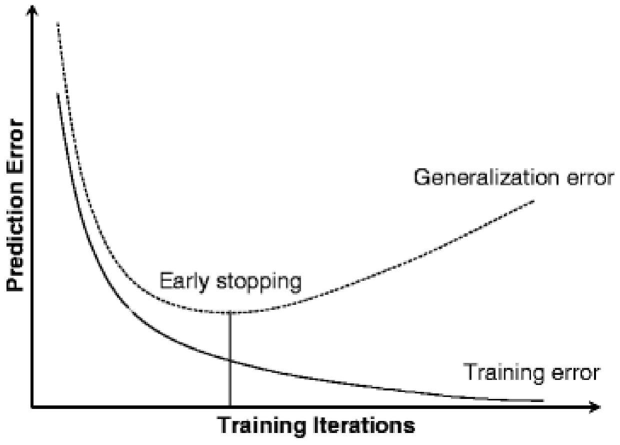


Figure 4.5 - Early stopping technique

However, the loss function can have a plateau, or increase slightly and then decrease again, causing a problem in the model, which is stopped when it is still learning. To avoid this, a kind of delay is inserted, a number of epochs in which it is allowed to accept not to see improvements, called *patience*.

This method is only applied to regressor, even if could be a good improvement for classifiers point of view in the next developments, to avoid overfitting and to speed up the computational time of the model.

4.4.3 Batch Normalization

Batch normalization technique was introduced by the author of the previous project, who performed an accurate study on this topic. This is not the aim of this work, but, since the model used for the new admissible classifier utilizes this optimization function, a brief explanation is needed.

Bath normalization is a technique used mainly for very complex deep neural networks. During the learning process, especially during backpropagation, the weights of one layer are updated based on the assumption that in the meantime the weights of the other layers do not change. However, this does not happen, because they are continuously updated as those of the previous layers change, leading to a slowdown in the process due to the attempt to coordinate all these changes. The main effect of this function is precisely to help this coordination, with the final result of speeding up and stabilizing the learning of the network.

This optimization is applied to hidden layers and output layer in both the two classifiers, while in the regressor is also applied to input layer.

4.5 Representation methods

One of the fundamental aspects of a neural network algorithm is to represent the results obtained, both to present them and to analyse its functioning, being, as mentioned above, mostly black box. The most used representations are those concerning the *learning curves*, in addition to the *confusion matrices*, which are also used to calculate different results. Then there are graphs related to *explainable machine learning*, with the aim of making the reasoning of the network more transparent. These have been presented in the next chapter.

4.5.1 Confusion matrix

After data acquisition, pre-processing, training and testing, it is important to find the results that certificate the accuracy and the effectiveness of the model. The better they are, the better the performance of the model, that this the aim of each neural network work. To compute some of the most important performance values a *confusion matrix* is needed.

It is only used for classification performance measurements and is a table composed of all the possible combinations of *predicted* and *actual values* of the classes considered in the classifier. In the case of this project, a binary classification is used in both the two classifiers, therefore the table is composed of four different combination: two for *predicted values* (positive 1 and negative 0) and two for *actual values* (positive 1 and negative 0), as it is shown in Figure 4.6.

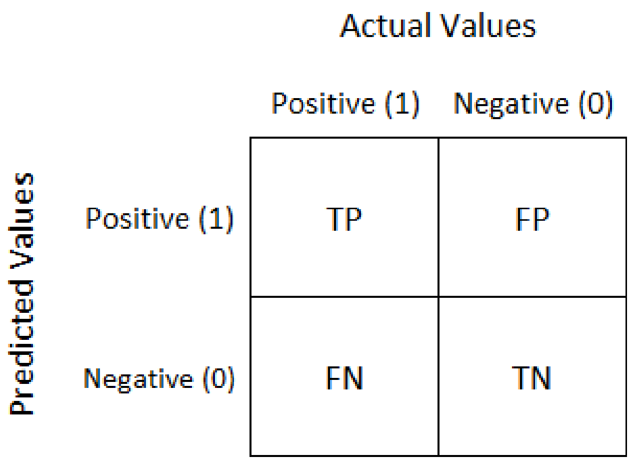


Figure 4.6 - Confusion matrix scheme

Figure 4.6 shows four different terms: True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN). The term *True Positive* means that the model has predicted positive and the actual value is positive. In the *False Positive*, on the other hand, the model predicts a positive sample that is actual negative; this is defined as a type 1 error. There is an error, this time of type 2, also in the *False Negative*, in which the model predicts a negative sample, but the actual value is positive. The last combination is that of *True Negative*, that is, the model correctly predicts that the sample is negative.

Thanks to this table performance indexes are calculated, like *accuracy, recall, F1_score, precision* and *Matthews Correlation Coefficient (MCC).* Although the computation of all these results, the author takes into account especially two of them: accuracy and MCC, that have been analysed in results chapter. For this reason, below only these two are explained.

- ***Accuracy***
  Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{Total}$$

  That means how many samples are predicted correctly out of all the four classes.

- ***Matthews Correlation Coefficient***
  MCC is defined as:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}$$

  This coefficient can be used only in binary classification problem.

Although *accuracy* is used more than *MCC*, the latter has some advantages in certain situations. For example, in the situation in which the dataset is unbalanced, i.e. the number of samples contained in one class is much greater than those contained in another, *accuracy* can no longer be considered a reliable measure as it overestimates the abilities of the classifier. On the contrary, *MCC* is not affected by the number of samples contained in the classes. [17]

4.5.2 Learning curves

These curves usually represent the improvement of the model along the epochs of training, therefore on the x-axis the number of epochs is shown, while on y-axis the learning improvements. These improvements can show by loss function, and so the smaller the better, like the *cross-entropy* or by performance index, like *MCC* or *accuracy*, in which the higher the better.

The improvement of the model in this plot can be evaluated on two different curves: the one found from training set or the one from validation set. The first gives the idea of how well the model is learning, while the second how well it generalizes the data. In this project these two curves have been represented together in the same plot.

Another important reason why learning curves are studied is to diagnose the behaviour of the model, in order to understand if this is subject to problems such as overfitting or underfitting. During the elaboration of the results of the pipeline, this aspect helped a lot the author of the project to understand which path was the right one, how to act to optimize the model.

If, for example, the training loss, at the end of the epochs of training, shows a continuous decrease of the value, this means that the model can still improve and that that number of epochs is too low. From this it can be understood that there is an underfit problem.

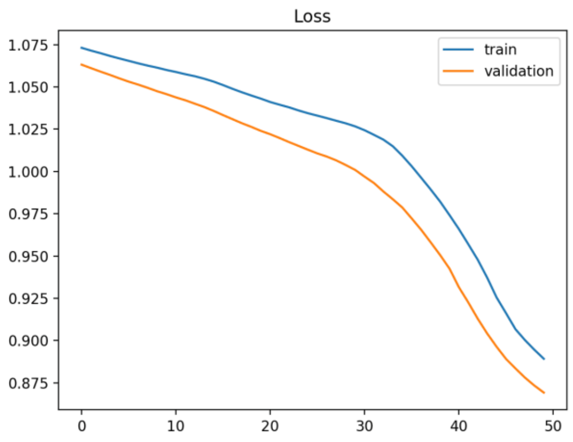


Figure 4.7 - Underfiting in learning curves

On the contrary, if the two curves of validation and training loss diverge with increasing epochs, with that of validation which, after reaching the minimum, starts to grow again, while that of training continues to decrease, a problem of overfitting is present.

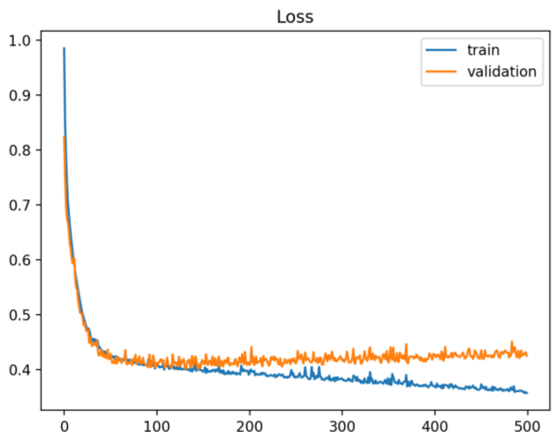


Figure 4.8 - Overfitting in learning curves

The trend of the curves that must be sought, i.e. a good fit the model on the data, is therefore represented by Figure 4.9

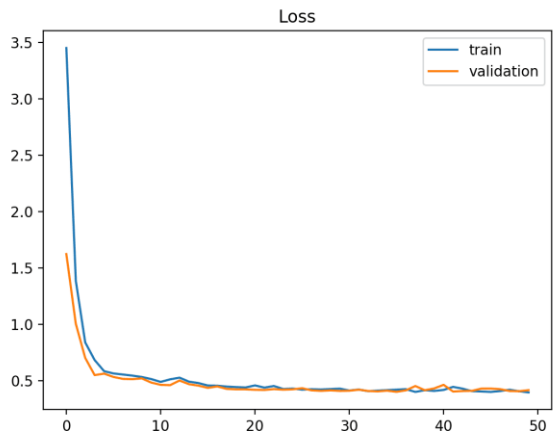


Figure 4.9 - Good fit in learning curves

Thanks to learning curves it is possible to understand also if the split of the training and validation set is correct. In fact, if training set has too few samples compared to validation set, the learning of the model can be insufficient. On the other hand, if validation set has too few samples, the model can’t generalize in the desired way. These two problems are shown in Figure 4.10. [18]

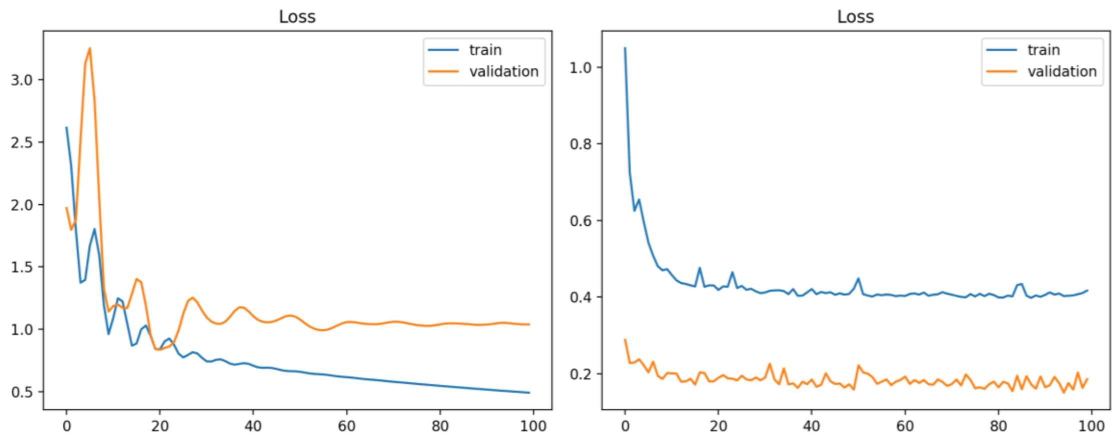


Figure 4.10 - Learning problem (left) and generalization problem (right) in learning curves

4.6 Permutation Feature Importance

In the previous chapter, a brief introduction to the topic concerning Explainable Artificial Intelligence has been presented, with an analysis of the various techniques that are used to make deep learning algorithms more transparent and understandable. Making an algorithm like the one used in this project completely explainable is almost impossible due to the huge number of variables, neurons, layers and functions used to write the algorithm and the model.

There are some techniques, however, that can provide some very useful information to increase understanding of the model, such as *feature importance*, which falls into the post-hoc explainability category. In particular, *permutation feature importance* has been analysed.

It is an inspection technique, applied to a trained model, which calculates the decrease in performance of the model when a feature is shuffled randomly a number k of times. it can be used both the training and testing datasets, depending on what kind of information it is needed: how much the model predictions depend on each feature or how much the model performance with unknown datasets depends on each feature, respectively. In this case the training set is used.

The algorithm that is used to compute the importance is the following:

- A reference score s is computed. In this case Matthews Correlation Coefficient is considered.
- Then, for each feature j:
    - For each iteration k in K total iterations:
        - The *j feature* column in dataset is permutated.
        - The new score $s_{k,j}$ is computed.
    - Finally, *permutation feature importance* i_j is found:

$$i_j = s - 1/K \sum_{k=1}^{K} s_{k,j}$$

There are several reasons why the author decided to implement a permutation feature importance algorithm instead of the classic feature importance. The first, more practical, is due to a greater compatibility between the pre-existing code and this algorithm. The model of the feasibility classifier developed in the previous project, and therefore also of the admissibility classifier, uses functions from the Python *Scikit Learn* library, the same used by the permutation feature importance. Therefore, the adoption of this technique has been inevitable.

The second reason is, instead, conceptual. The permutation feature importance uses, as mentioned above, a model trained once. Contrarily to what happens in the classic feature importance technique, where the model is trained every time a feature importance is calculated. This difference obviously leads to a higher computation time.

Furthermore, the feature importance procedure performs the calculation of the score by completely eliminating one feature at a time, not just permutating their values. This mechanism causes the model to be trained on a different reduced dataset each time, altering the results [19].

If a graphical representation of the results is needed, a possible solution is the one represented in Figure 4.11.

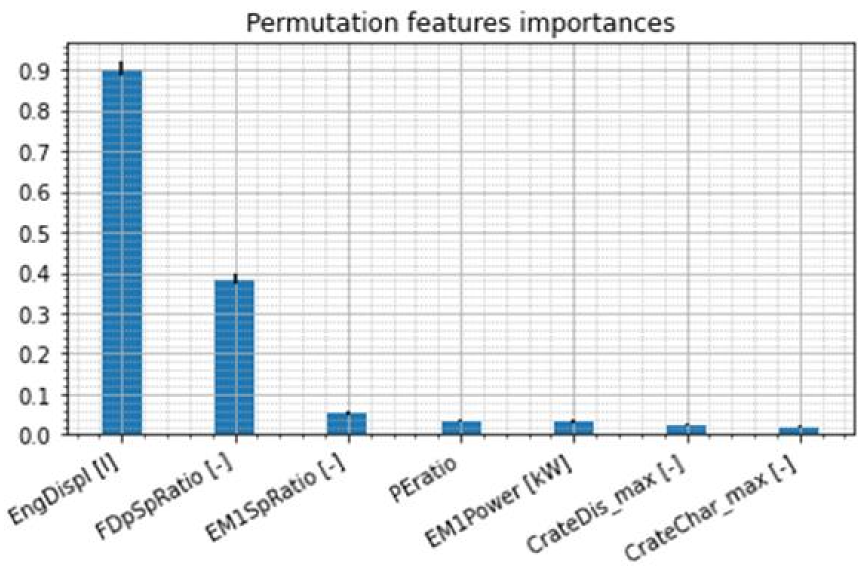


Figure 4.11 - Permutation feature importances of admissibility classifier

The plot shows an example of permutation feature importance of the admissibility classifier. It is clear that the taller is the blue column, the more important is the correspondent feature. This is explained by the formula above; in fact, shuffling the column of a feature, if it is very important for the predictions of the model, the score will decrease consistently. Otherwise, if the score remains more or less the same, the difference between the two term of the formula will be small, as will the importance of the feature.

From the chapter concerning the description of the datasets characteristics it is known how the number of features is eight, since the feature linked to the type of cycle on which the considered layout is simulated has been added to the seven already present. However, if the number of features represented in the Figure 4.11 is counted, it is possible to see that the number is 7, the 'cycle' feature is missing. This is because only the features that meet the following condition are plotted:

$$importances - 2 * std > 0$$

Where *std* is the standard deviation of the measure, since the multiple scores computed have a variation. Standard deviation is represented by the vertical black segment on th top of the blue column of the plot.

Furthermore, if the sum of the value of these importances is calculated and analyzed, it could be seen that this is not equal to one. This is because the ordinate axis does not represent a percentage, but the absolute decrement of the reference score due to the permutation of the considered feature.

5 Results

This chapter, relating to the presentation of the results and the procedure that led to them, is divided into three main parts.

Initially the project developed with the design of the admissibility classifier and its subsequent integration into the pre-existing pipeline, using the P2 and P4 datasets of the old project. The first part therefore is about the analysis of the new classifier and the changes that this insertion has caused within the entire pipeline.

In the second part, however, the focus is shifted to the introduction of a new P2 dataset expanded compared to before, analysing the changes made to the network.

The third and last section is an in-depth analysis of the permutation feature importance, in which the most important features for the two classifiers will be evaluated.

5.1 New admissibility classifier in the pipeline

5.1.1 Analysis on P4

Starting from the feasibility classifier, the admissibility classifier was created. As already explained, it has been inserted into the pipeline with the possibility of considering it or not through an ON / OFF button.

Furthermore, for all the analyses in this section, the P2 and P4 datasets already used in the previous project were used, therefore containing 1500 layouts simulated on the WHVC cycle.

The first study presented is the one concerning the P4 architecture. The Figure 5.1 compares the performance of the two classifiers, namely the Accuracy and the Matthews Correlation Coefficient.

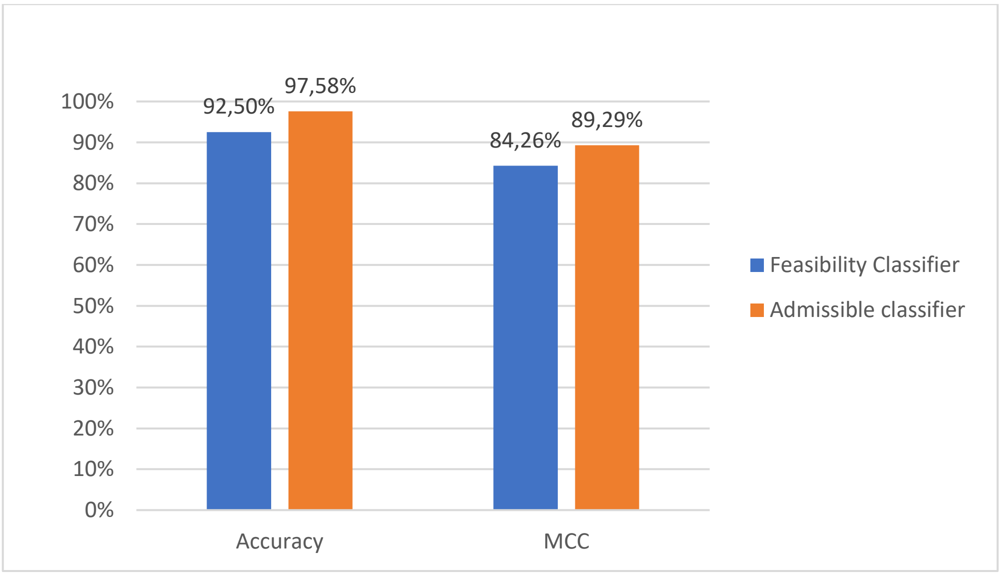


Figure 5.1 - Feasibility and admissibility classifier performances (P4)

As can be seen from a very first analysis, the performance of the admissibility classifier is good, even better than that of the first classifier. This may be due precisely to the action of the latter, which by carrying out a first filter on unfeasible layouts, makes easier the second classification.

However, it must be borne in mind that most of the results shown are the result of an average of results deriving from some simulations, usually between four and seven. Such a low number, essentially due to a fairly high simulation time, means that that the measurements are subject to sometimes high standard deviations, thus not leading to relevant behaviours and trends.

After verifying the stability of the new network, the attention is extended to the entire pipeline, and a comparison is made between the performance of the regressor with and without the insertion of the second classifier. On the left of Figure 5.2 is shown the percentage value of the R2 index, while on the right the value of the considered loss function, RMSE.

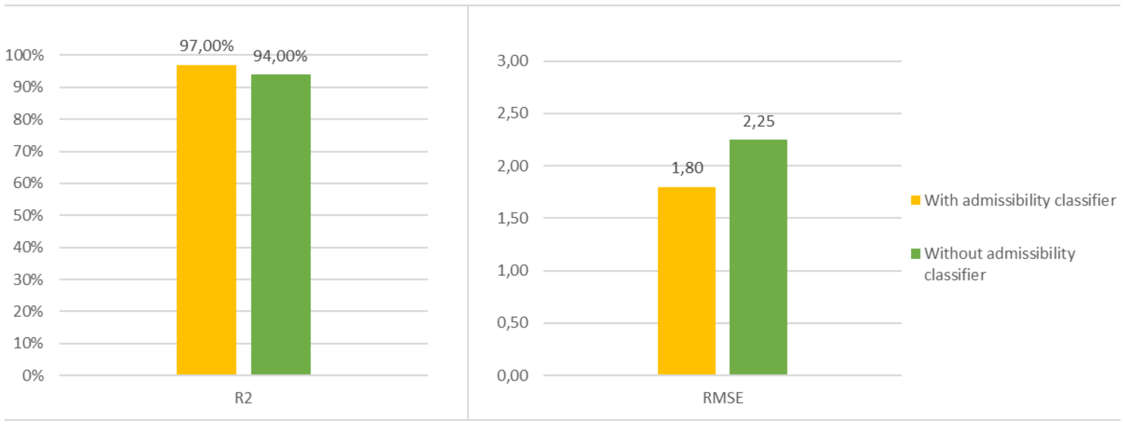


Figure 5.2 - Regressor with & without admissibility classifier (P4)

The graphs show how the regressor positioned at the end of the pipeline has improvements with the addition of the admissibility filter. In fact, R2, the less significant value between the two for the reasons seen in the previous chapters, increases by 3%, while the RMSE has a decrease, therefore an improvement of 25%.

However, looking at the results of each simulation and their standard deviation in Table 5.1, it can be seen that there is no clear improvement in the network.

Table 5.1 - Regressor with and without admissibility classifier

	With admissibility		Without admissibility	
	Loss Testing	R Testing	Loss Testing	R Testing
Simulazione I	1,46	0,98	2,57	0,93
Simulazione II	2,30	0,94	2,18	0,94
Simulazione III	1,10	0,99	2,37	0,92
Simulazione IV	2,32	0,95	1,86	0,95
Average	1,80	0,97	2,25	0,94
Std	0,61	0,02	0,30	0,01

This uncertainty is essentially due to two main reasons. The first is the limited number of simulations; the second, on the other hand, has a high percentage of network randomness. This in fact depends on the choice of hyperparameters that occurs from a random search. Also, the seed is not locked. The meaning of seed is the following: any random number using a random number generator. A seed is used to generate the sequence of numbers, usually linked to the current time in milliseconds, so that each time this generator is different, always making the sequence of numbers created different. If you want, on the contrary, to make these sequences more similar, you can set a specific number to the generator, seed [20]. To solve as much as possible the problem of randomness, in the analysis done on P2 architecture the seed has been fixed and a fixed set of hyperparameter has been chosen, as it has been explained below.

Analysing also the *learning curves* of the three networks, it can be seen that the *cross-entropy* graph of the feasibility classifier, Figure 5.3 , presents a tendency to overfitting, since the mean validation and training curves diverge as the epochs increase.

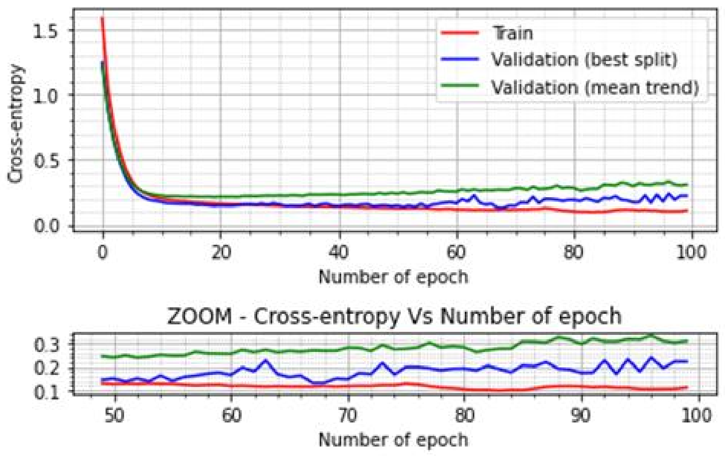


Figure 5.3 - Cross-entropy vs Epochs - Feasibility classifier – without Dropout (Sim I)

As seen above, a possible solution to this problem may be to add a dropout layer to the classifier model, since it is absent until now, obtaining an improvement, as shown in Figure 5.4.

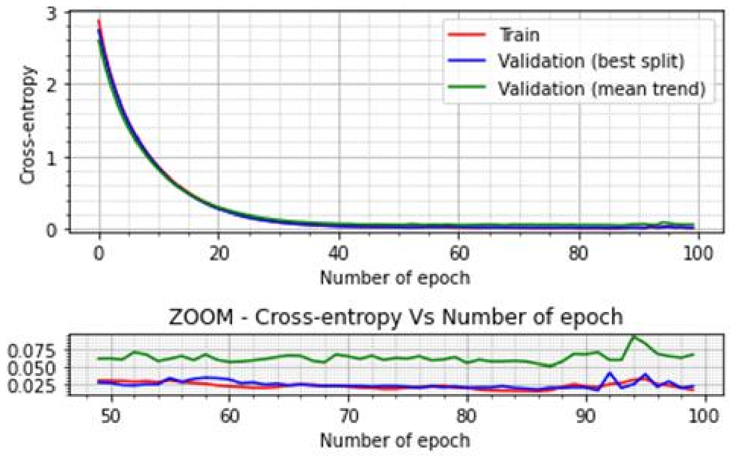


Figure 5.4 - Cross-entropy vs Epochs - Feasibility classifier – with Dropout

5.1.2 Analysis on P2

The same analysis previously carried out for the P4 hybrid vehicle architectures is now carried out for the P2 architectures. Also in this case, in the first simulations the unlocked seed was kept, with not very convincing results and, above all, not very comparable, especially in terms of standard deviation value. For this reason, it was decided to carry out a series of simulations with seeds locked at 7 and with a fixed combination of hyperparameters of the three networks, defined by evaluating those chosen in the best simulations. The Table 5.2 shows the values.

Table 5.2 - Fixed hyperparameters

FEASIBILITY CLASSIFIER	
Hidden layers	1
Neuron first	76
Batch size	40
Learning rate	0.00147
L2 regularization	0.0213
ADMISSIBILITY CLASSIFIER	
Hidden layers	3
Neuron first	134
Batch size	21
Learning rate	0.00215
L2 regularization	0.00206
REGRESSOR	
Hidden layers	4
Neuron first	44
Batch size	52
Learning rate	0.008324326
Weight initialization	Glorot uniform
Dropout	0

The first results shown in Figure 5.5 concern the performance of the two classifiers. In this case the standard deviations are small, as the black vertical segment show, and this demonstrates the greater stability of the results under these conditions. Furthermore, it can be noted that the average values of the seven simulations carried out are higher now, considering P2 architectures, compared to before in which P4 architectures were considered.

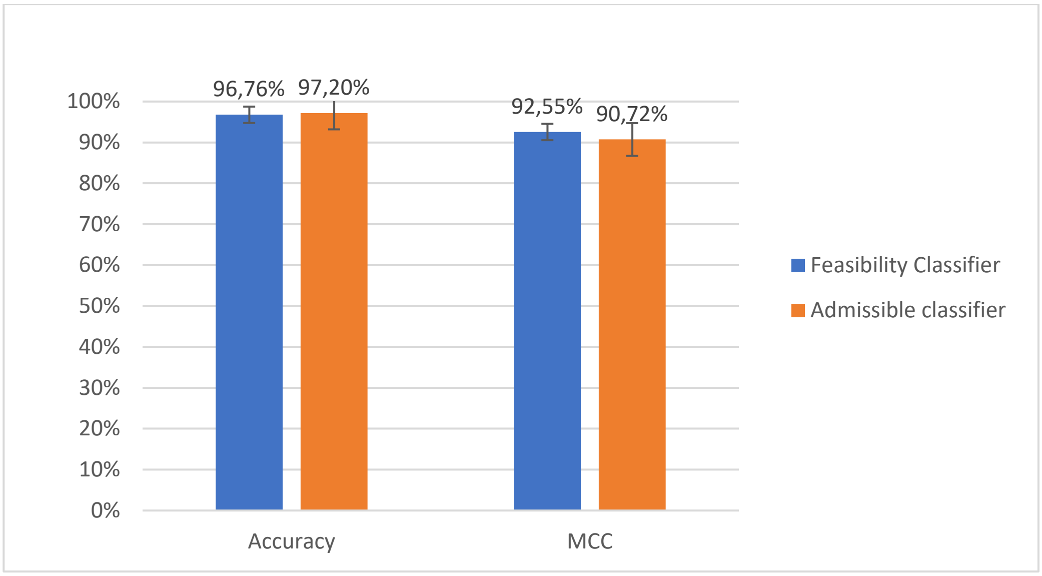


Figure 5.5 - Feasibility and admissibility classifier performances – Fixed parameters (P2)

Turning, instead, to the regressor (Figure 5.6), the important result is that the improvement of RMSE with the introduction of the new classifier is of the same order of magnitude found previously, about 28%. This underlines again and more strongly the positivity of introducing an admissibility filter in the pipeline.

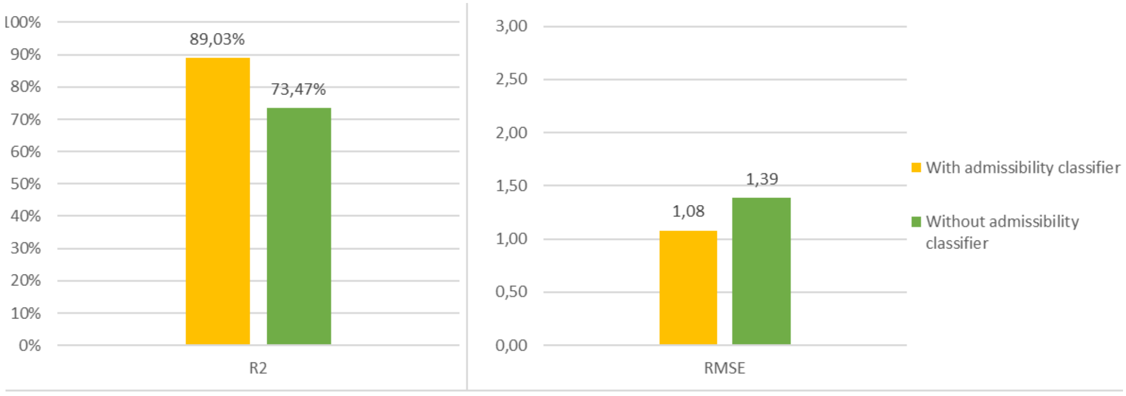


Figure 5.6 - Regressor with & without admissibility classifier – Fixed parameters (P2)

Blocking the seed helped the author to obtain more comparable results and to confirm the improvement of the pipeline. This process is usually used to isolate the causes resulting from a single change. However, the high randomness of the network is a fundamental aspect in the generalization capacity of the model. For this reason, the seed and hyperparameter search has been unlocked from now on.

Studying the learning curves again, it can be seen from Figure 5.7, in which the cross-entropy of one of the simulations is represented, how there is a tendency towards overfitting also with regard to this P2 architecture.

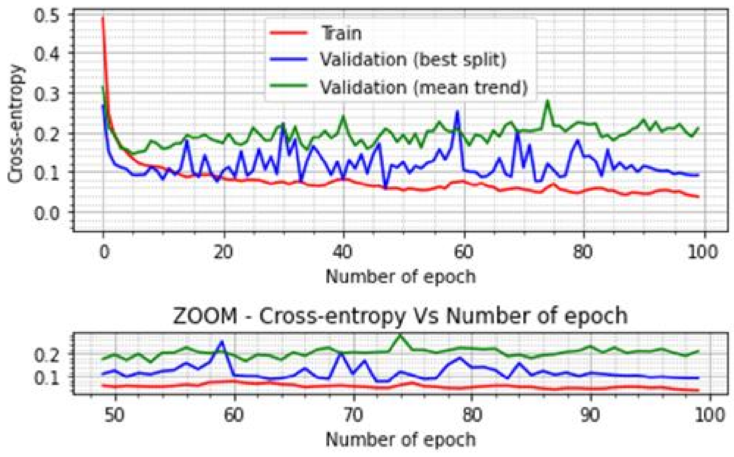


Figure 5.7 - Cross-entropy vs Epochs - Overfitting (P2)

To solve this problem, a different strategy with respect to P4 architecture is adopted. A study on the number of iterations adopted by the model is made, taking into consideration three values: 10, 20, 35 iterations. As explained above, the number of iterations corresponds to how many batches needed to complete an epoch. By increasing this number, model training should be more accurate.

However, evaluating the Figure 5.8, it is clear that there is no real tendency of improvement

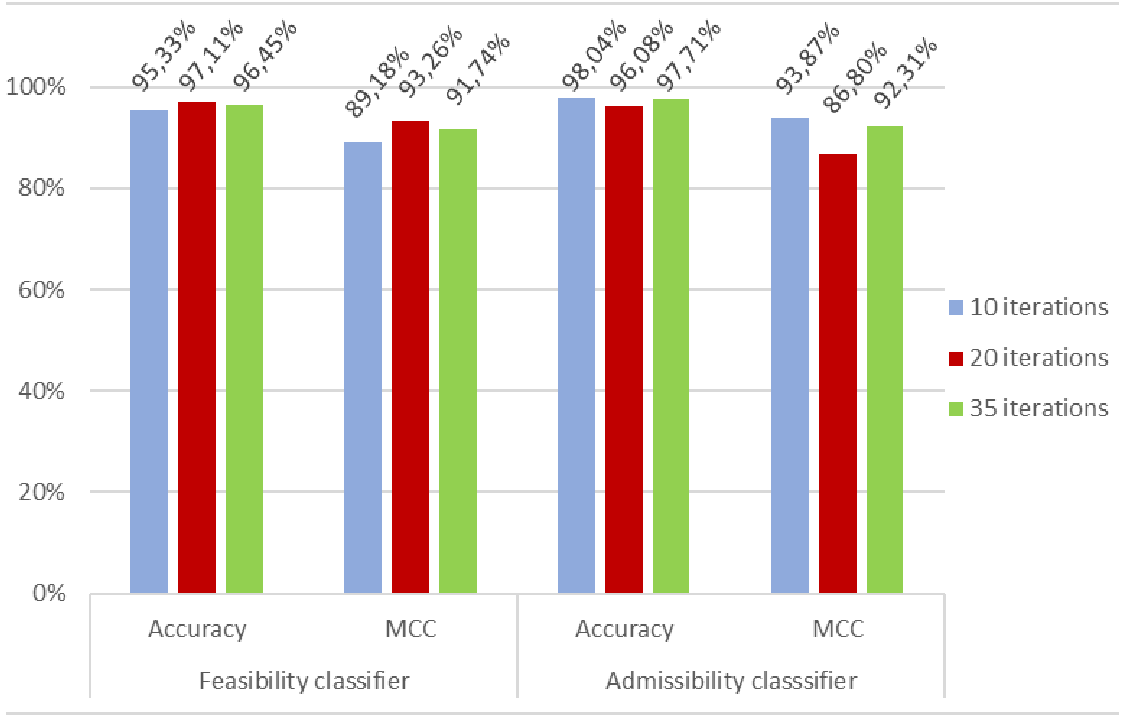


Figure 5.8 - Comparison between different iterations

On the other hand, overfitting improves slightly when simulations with 35 iterations are considered, but if the zoom present in Figure 5.9 is evaluated, there is still divergence.

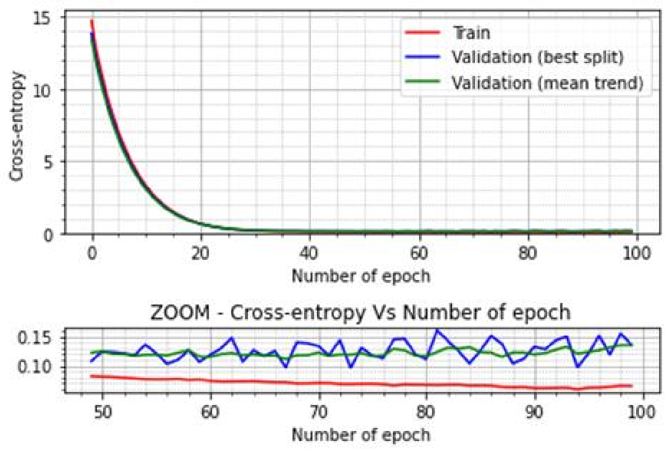


Figure 5.9 - Cross-entropy vs Epochs - 35 iterations

Furthermore, the significant increase in simulation time with iterations means that this parameter change is not entirely beneficial.

For these reasons the author has decided to set the iterations to 10, keeping a good simulation time, and insert dropout layers to improve overfitting. A simulation example is shown in

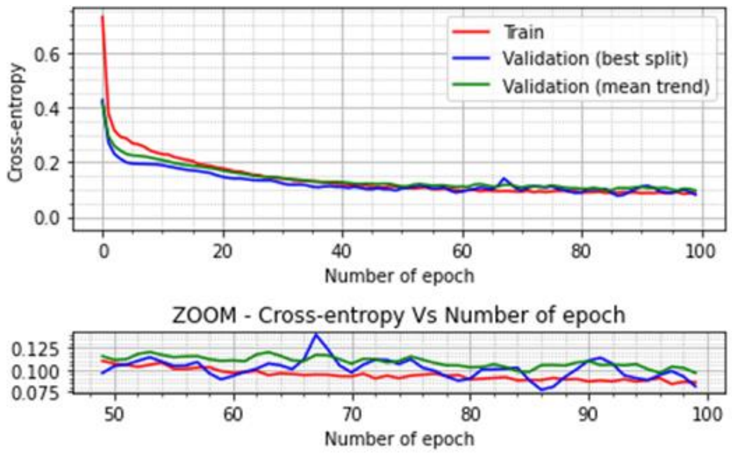


Figure 5.10 - Cross-entropy vs Epochs - with Dropout (P2)

5.1.3 *Comparison between P2 and P4*

This comparison was already faced in the previous project regarding the feasbility classifier. In this section we will study the performance differences of the admissibility classifier between the P2 and P4 architectures.

As can be seen in Figure 5.11, the performances of P2, with the same model, are better, especially for what concern the Matthews Correlation Coefficient. Obviously, there are some different parameters, being random, but the structure of the model and the fixed parameters are the same.

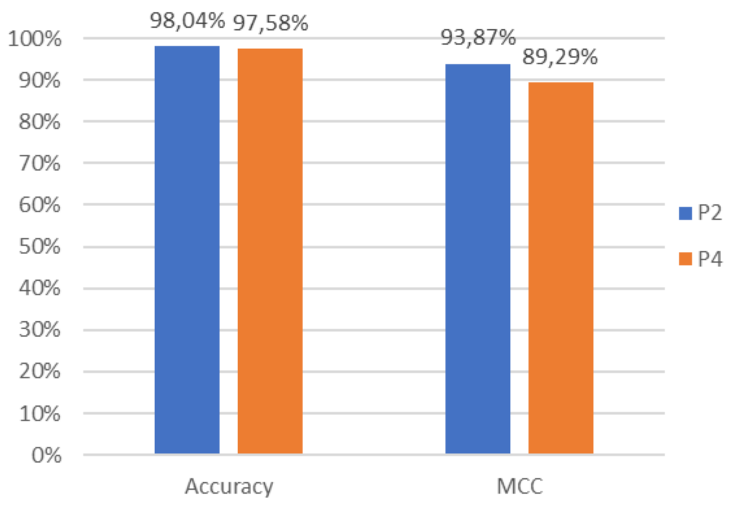


Figure 5.11 - Comparison between P2 and P4 admissibility classifier performance

This difference may be the result of a structural difference between the two architectures. If the design features of the layouts of P2, and also P3, is analysed, a parameter called "EM1SpRatio" (the speed-coupling device transmission ratio) is found. In this case, the electric motor is connected to the transmission system which exerts power only on the front axle. If, on the other hand, we consider P4, we find "FDsSpRatio" (transmission ratio at the secondary axle differential), which describes the connection of the electric motor on the rear axle. This decoupling of the two motors means that there are two transmission ratios and, therefore, more losses, compared to a single one, as happens in P2 and P3. This is a more physical explanation.

Studying the dataset, and in particular the connection between performance index value and the two features presented above, the graphs in Figure 5.12 and are created.

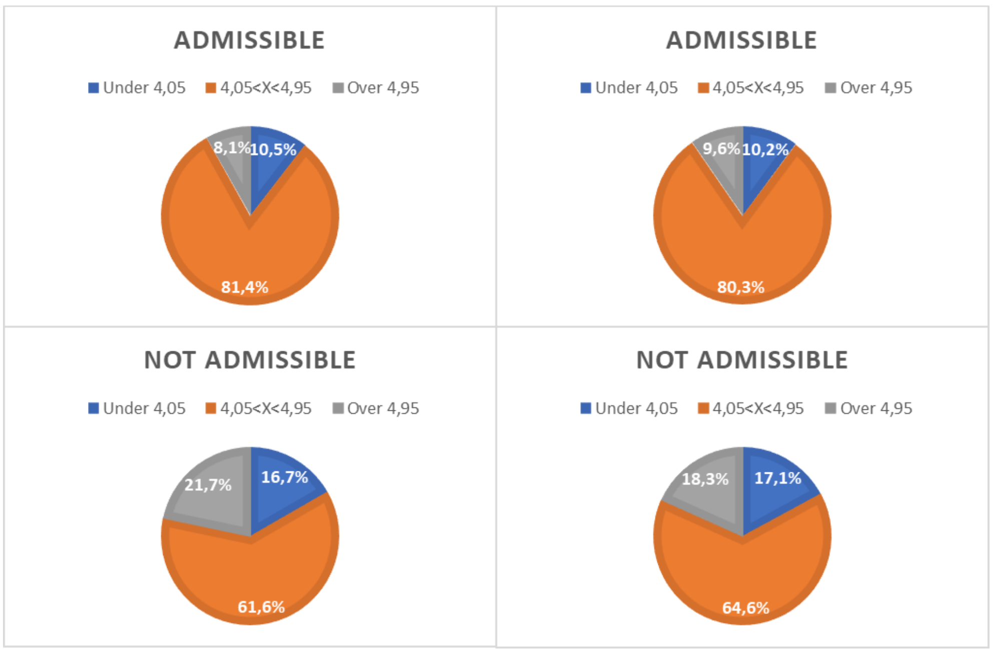


Figure 5.12 - EM1SpRatio feature in P3 dataset (left) and P4 dataset (right)

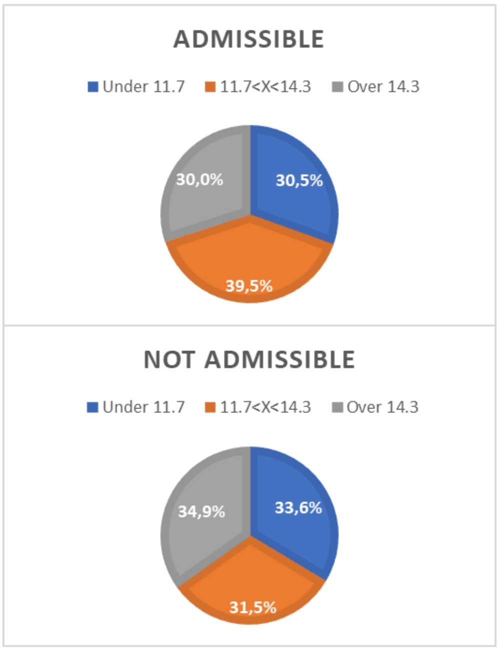


Figure 5.13 - FDsSpRatio feature in P4 dataset

As also for feasibility, also in admissibility there is a greater characterization of the dataset by "EM1SpRatio", which therefore helps the model in the classification. This can be the reason that causes higher performance of P2 classifier compare to P4 one.

5.2 Pipeline analysis with the new P2 dataset

This new section analyses the performance of the new classifier and the complete pipeline with the introduction of a new dataset. This, as fully explained, is a dataset expanded by five times compared to the previous one, in fact it contains 7500 layouts. To create it, five datasets containing 1500 layouts each similar each on a different cycle were merged. This made it possible to insert a new feature, the one of the cycle types.

Initially a pipeline that takes into consideration only the feasibility classifier is presented, and subsequently, the admissibility is inserted.

5.2.1 Feasibility classifier + regressor pipeline

As shown in the Figure 5.14, the performance of the feasibility classifier slightly increases, especially the MCC value. The interesting aspect to underline is the decrease in the standard deviation value, represented by the vertical segments. As for all graphs, in fact, the averages of several simulations are represented, in this case five. This demonstrates how expanding the dataset also increases the stability of the results, making them more reliable and comparable.

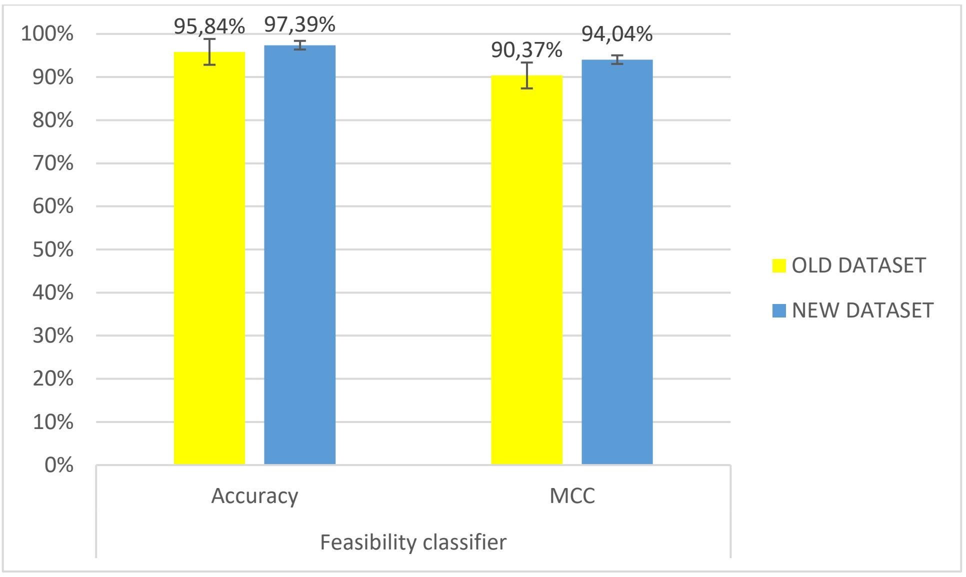


Figure 5.14 - Feasibility classifier performance - New P2 dataset

Conversely, comparing the regressor results shows a fairly large difference between the old dataset and the new one. Accuracy, in fact, grows a lot, but so does error, RMSE. However, it must be emphasized that the error remains more than acceptable, being 3.8 g/km on values of the order of magnitude of 300/400 g/km.

In any case, it is important to analyse the reason for these marked changes. These are probably due to the technique of calculating the two variables R2 and RMSE. As explained above, in fact, R2 depends on the difference between the mean of the values and the function of the model. Obviously, the more values there are, the more accurate the average will be and will approach the model function, increasing the R2.

Conversely, the RMSE depends on the distance of each value of the function curve. It is clear that increasing the number of values the number of distances considered increases, as well as the RMSE value.

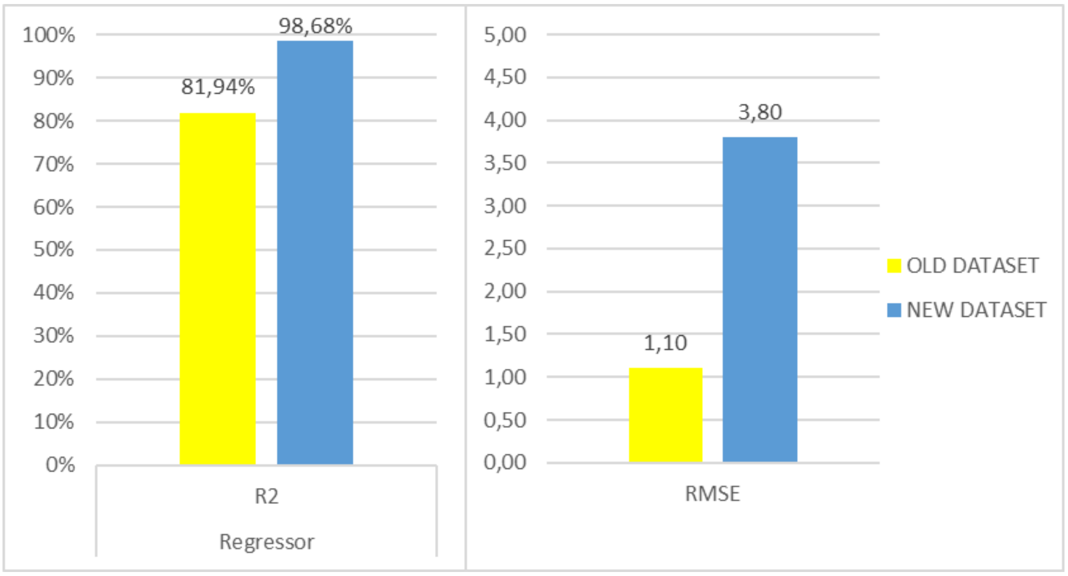


Figure 5.15 - Regressor performance - New P2 dataset

5.2.2 *Feasibility classifier + admissible classifier + regressor pipeline*

By inserting the new classifier now, the following results are obtained. Regarding the performance of the classifiers, as seen in Figure 5.16, there are no particular differences between the results derived from the two dataset, considering the slight variability of each simulation. All the results show very high performance of the classifiers.

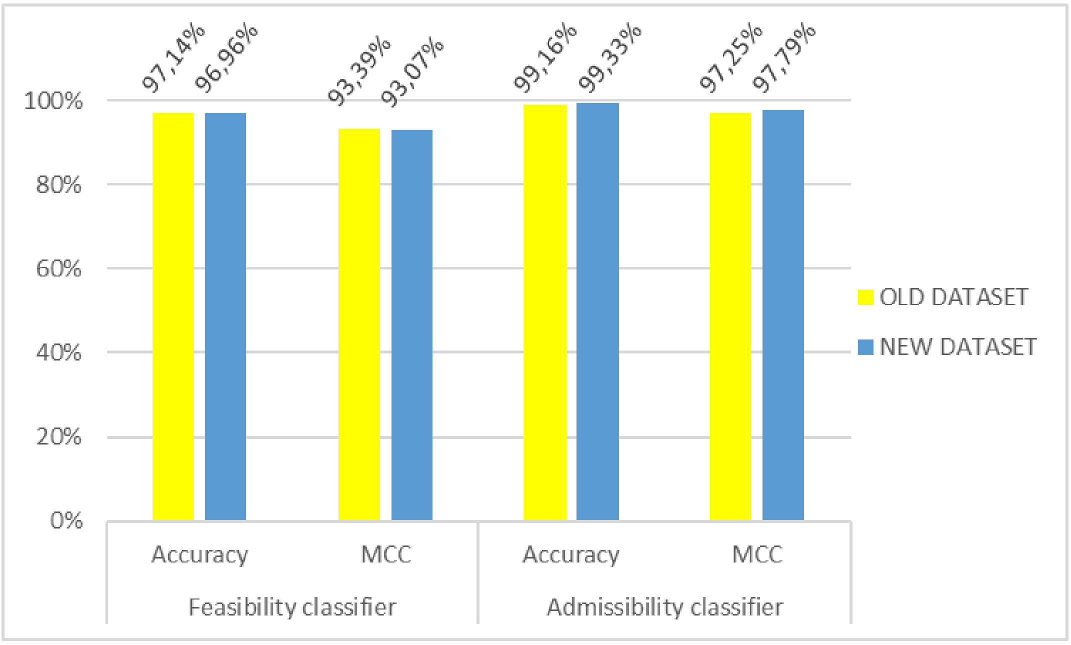


Figure 5.16 - Classifiers performances - New P2 dataset

Speaking instead of the regressor, Figure 5.17, the trends are the same as previously found in the case with only one classifier.

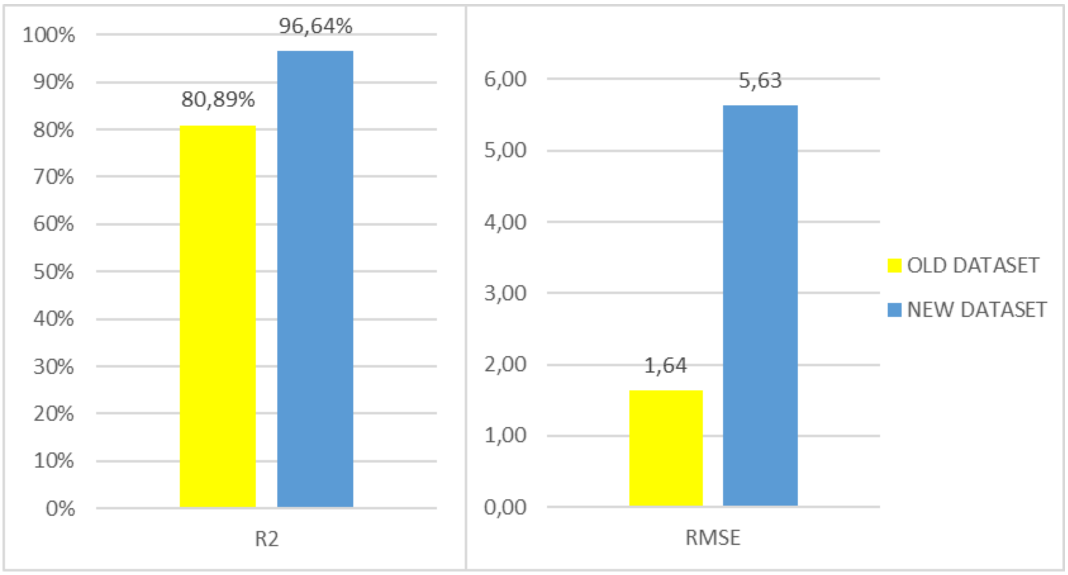


Figure 5.17 - Regressor performance - with admissibility classifier - New P2 dataset

The model used to obtain all the results related to the new dataset contains the dropout layers deriving from the overfitting study carried out previously. Applied to the new dataset, however, it leads to behavioural problems of the learning curves, as can be seen in Figure 5.18 and Figure 5.19.

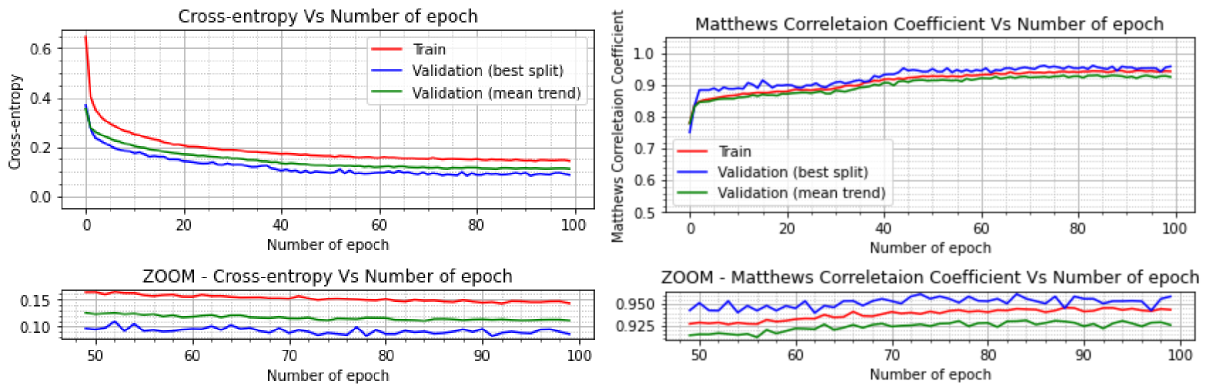


Figure 5.18 - Feasibility classifier learning curves

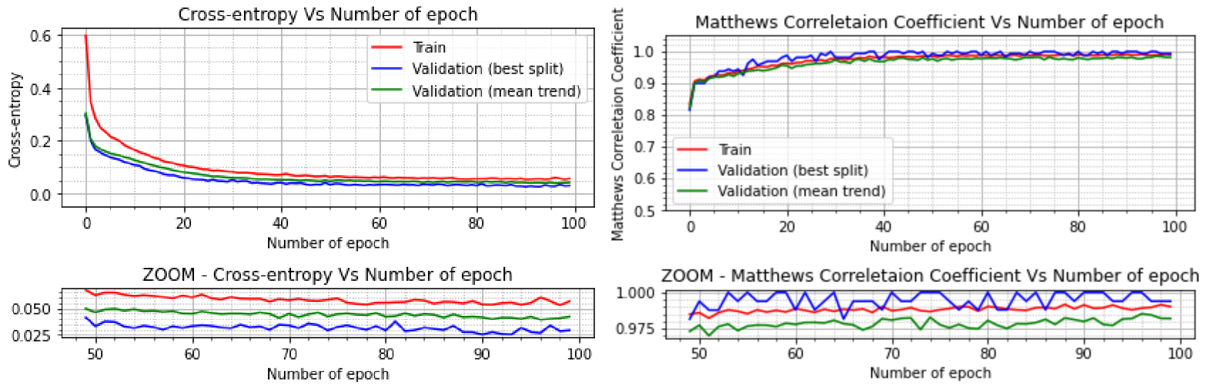


Figure 5.19 - Admissibility classifier learning curves

It can be observed that the train set curve is lower than the validation one in the MCC graphs and higher in the cross-entropy ones. This behaviour denotes better learning of the validation set than the training set, which shouldn't happen since the training curves should be the best in terms of performances.

This problem could be caused by too high a dropout rate. As previously explained, in fact, the dropout value indicates the probability that a node of the training set is turned off. If this probability is too high, there is a risk of weakening the training set too much, causing a problem like the one shown in the figure.

To limit this effect, the strategy adopted is to insert the dropout as a hyperparameter also in the two classifiers, letting the model choose the best value among the following ones directly through hyperparameter tuning: [0, 0.1, 0.2, 0.3].

In addition, to help the model in the classification, a deepening on the range of hyperparameters chosen by the algorithm in the best simulations is performed, obtaining a narrowing of the search space as follows in Table 5.3.

Table 5.3 - Restricted hyperparameter search space

FEASIBILITY CLASSIFIER	
Hidden layers	2 – 4
Neuron first	30 – 140
Batch size	32 – 256
Learning rate	0.0001 – 0.005
L2 regularization	0.0001 – 0.1
Dropout	0 – 0.3
ADMISSIBILITY CLASSIFIER	
Hidden layers	1 – 3
Neuron first	100 – 200
Batch size	32 – 128
Learning rate	0.0001 – 0.01
L2 regularization	0.0001 – 0.001
Dropout	0 – 0.3

Table 5.4 shows the results of the simulations carried out with the dropout inserted as a hyperparameter and the search space restricted; the column of the dropout values is also inserted, to monitor it. There are some important considerations to make.

Table 5.4 - Pipeline simulations - Dropout as hyperparameter & restricted search space

	FEASIBILITY CLASSIFIER			ADMISSIBILITY CLASSIFIER			REGRESSOR	
	Accuracy	MCC	Dropout	Accuracy	MCC	Dropout	Loss Testing	R Testing
Simulation 1	96,27%	91,45%	0,0	99,60%	98,72%	0,2	2,06	99,70%
Simulation 2	97,07%	93,40%	0,2	99,80%	99,35%	0,1	5,22	97,60%
Simulation 3	97,07%	93,33%	0,1	99,80%	99,35%	0,1	2,92	99,36%
Simulation 4	95,33%	89,36%	0,0	99,40%	98,06%	0,2	1,75	99,77%
Simulation 5	96,40%	91,85%	0,0	99,40%	98,06%	0,1	1,96	99,60%
Average	96,43%	91,88%	0,06	99,60%	98,71%	0,14	2,78	99,21%
Std	0,01	0,02	0,09	0,00	0,01	0,05	1,43	0,01

For the first, it is necessary to observe the learning curves of the next figures, in which simulation 2 and 5 are taken as reference.

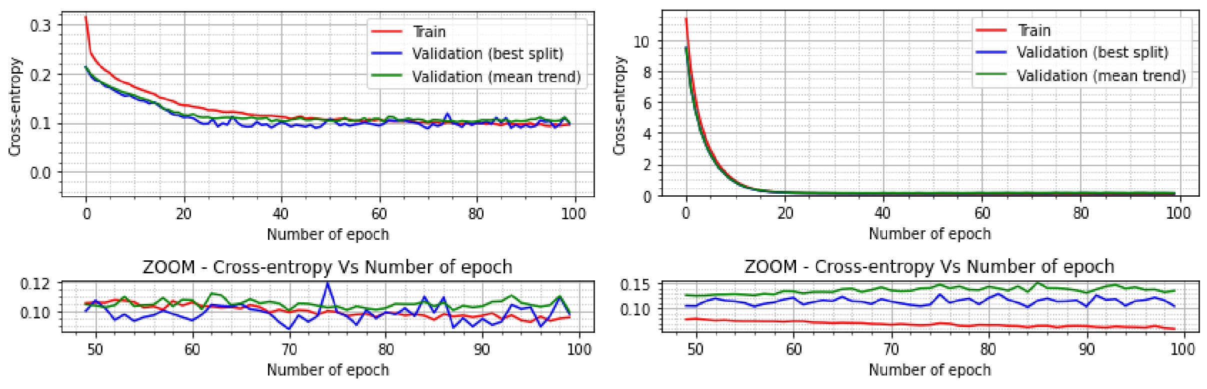


Figure 5.20 - Cross-entropy vs Epochs - Feasibility classifier - sim 2 (left) & 5 (right)

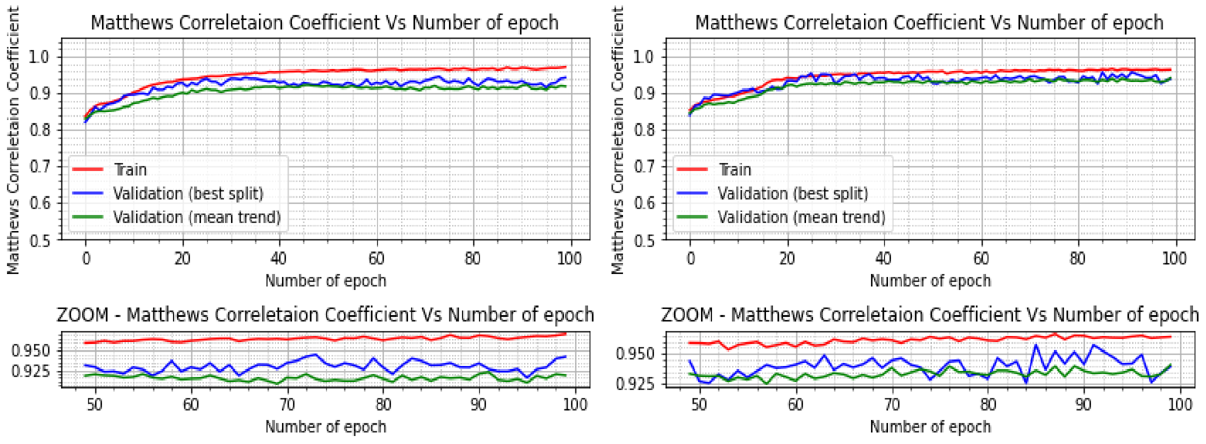


Figure 5.21 - MCC vs Epochs - Feasibility classifier - sim 2 (left) & 5 (right)

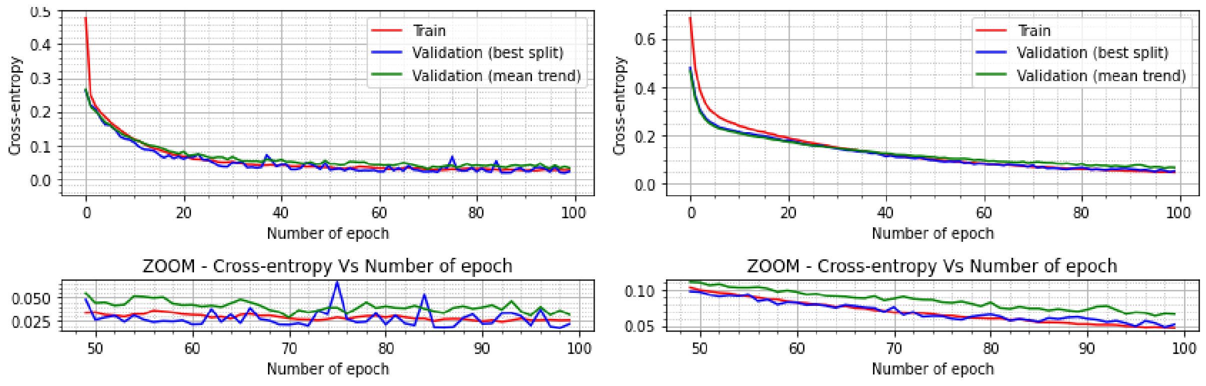


Figure 5.22 - Cross-entropy vs Epochs - Admissibility classifier - sim 2 (left) & 5 (right)

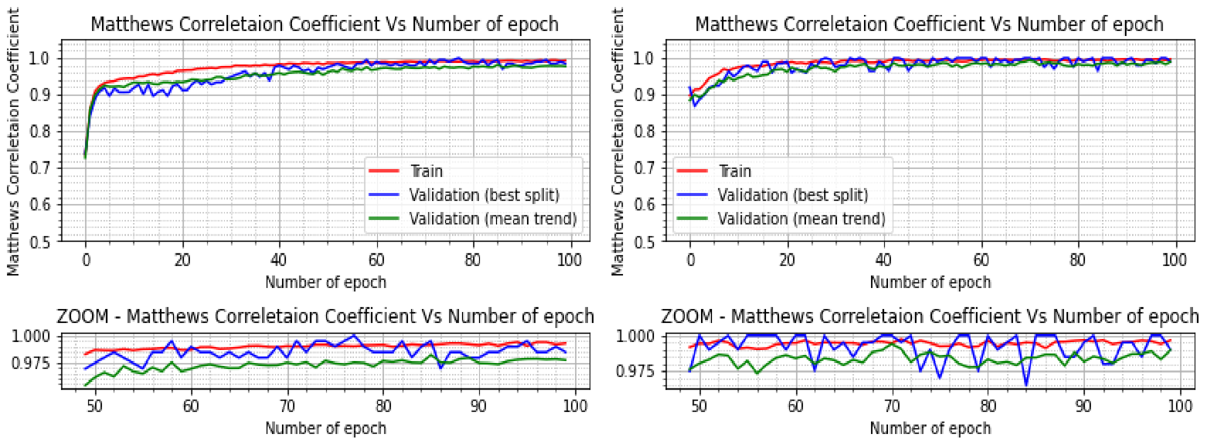


Figure 5.23 - MCC vs Epochs - Admissibility classifier - sim 2 (left) & 5 (right)

The previous graphs represent simulation 2 (on the left) and 5 (on the right) of Table 5.4. In the first the code has chosen a higher feasibility classifier dropout, that is 0.2, while in the second it is equal to 0.1. In the model with higher dropout, we can observe the problem of having the training set with worse performances compared to the validation set, already found previously with higher dropout. However, this model has no tendency to overfitting.

Moving on to the simulation with a lower dropout, however, we see a more suitable behaviour of the train and validation curves, closer to the desired and expected ones. Unlike the previous model, however, looking at both Figure 5.20 on the left and Figure 5.21 on the left, there is a slight tendency to overfitting. This behaviour had already been found in previous studies, when the dropout layer was introduced to eliminate it.

Both Figure 5.21 and Figure 5.23, where MCC are represented, show a consistent improvement with respect to those with greater dropout.

Observing the graphs of the other simulations, it can be seen that there is a very positive trend regarding the behaviour of the train and validation curves as the dropout percentage decreases, while there is not a well-defined trend if overfitting is considered. For this reason, according to the author, a good solution could be to use a low dropout value as the hyperparameter chosen by the code, as has already been set, and insert an *early stopping* in the classifiers, as also happens in the regressor, so that model training is stopped in case of overfitting. Furthermore, this element could speed up the simulations since the model takes, most of the time, a few epochs to reach the minimum.

The second consideration regarding this analysis is that, observing the average values of the five simulations, there is a significant decrease in the RMSE of the regressor, "Loss Testing", compared to the previous values (Figure 5.24), despite having a non-negligible standard deviation.

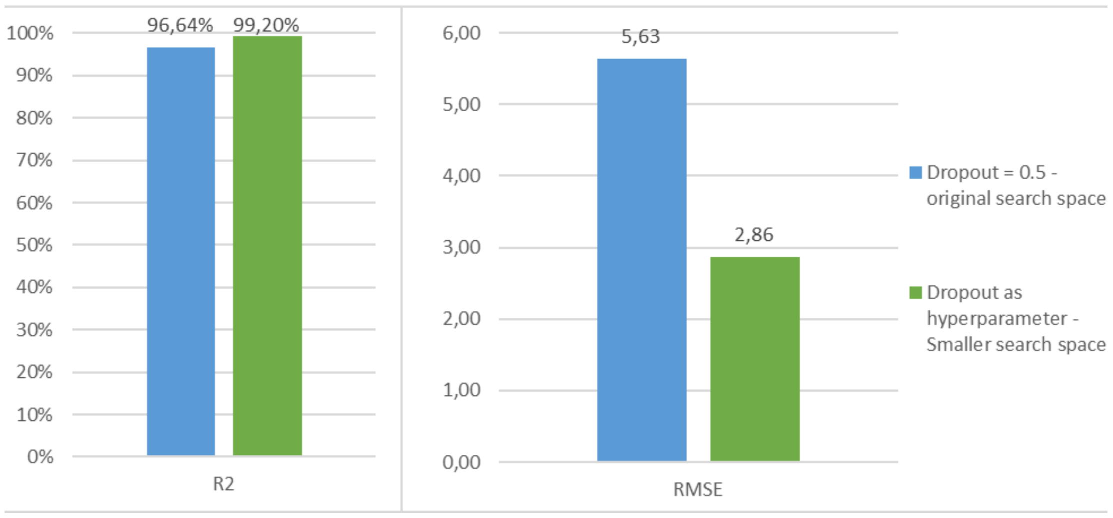


Figure 5.24 - Regressor performance with different parameter combinations

5.3 Permutation Feature Importance results

Permutation feature importance is, as explained above, a technique for increasing the transparency of the neural network model. In this case it was implemented for the two classifiers.

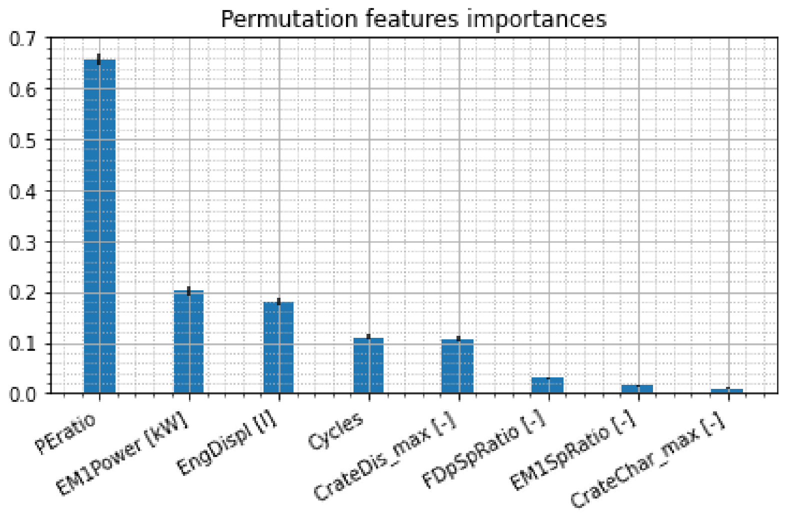


Figure 5.25 - Permutation Feature Importance for feasibility classifier

As can be seen in the Figure 5.25, this technique is used for the model that refers to the new P2 dataset, as there is the "Cycle" feature in the list, which indicates on which cycle the considered layout is simulated. It is noted how the most important features regarding the feasibility classification are the "PEratio", the "EM1Power" and the "EngDispl", with a good importance also of "Cycle" and "CrateDis_max". This last good dependence of the classifier performance on the new "Cycle" feature certifies the effectiveness of this method. The type of cycle, in fact, certainly affects the amount of CO2 that a layout can emit.

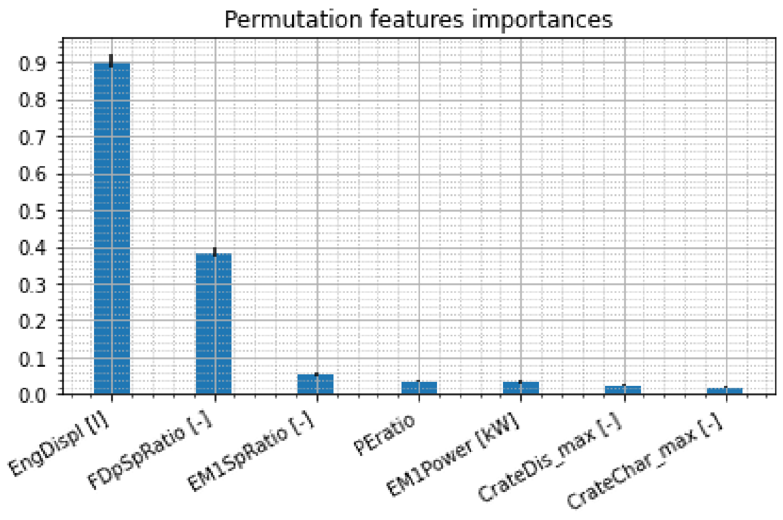


Figure 5.26 - Permutation Feature Importance for admissibility classifier

On the contrary, if Figure 5.26 is taken into consideration, it can be seen that this feature is not even represented, due to the presence of a constraint that ‘filters’ the less important features. This result is totally reliable, as the type of simulation cycle does not affect the performance of a vehicle. This depends on the characteristics of the engine, and, in fact, the displacement of the engine is considered of extraordinary importance, as well as the final drive speed ratio.

Observing both graphs, it can be seen that the two classifiers depend on different features, as expected, as they refer to different objectives. This is important to consider if a future task is to explore this topic further. One of its developments could be to introduce what is called *relative feature importance*, in which the features that are not considered important are eliminated in the continuation of the pipeline. From this brief explanation it is clear how the dependence of the two classifiers on different features is a delicate situation. If, in fact, a feature not important for the first classifier, were to be important for the second network, its elimination could lead to a serious deterioration of the performance of the second network.

Particular attention must therefore be paid to this aspect.

Conclusions

To sum up, this project started from a pre-existing pipeline, created to support and integrate a Dynamic Programming algorithm aimed at calculating CO2 emissions of hybrid vehicles of different architectures, P2, P3 and P4.

In particular, the task of this project was to add another classification network, an admissibility classifier, within a pipeline already composed of a feasibility classifier and a CO2 regressor.

The project focused especially on P2 architecture layouts, with the introduction of a new, expanded dataset of this type.

The focus of the work was to verify if the introduction of this new classifier was good for the entire pipeline, and if the model improved, as is widely explained in the literature, with a higher number of layouts.

As shown in the results chapter, all of these analyses led to very positive results, with high performances of all the nets.

In parallel, an in-depth research was also performed on XAI, with a particular focus on a post-hoc explainability technique, the *permutation feature importance*.

Possible future developments could concern various aspects. Firstly, further analysis on the influence of the dropout percentage adopted by classifiers could be indispensable for its correct use.

Furthermore, as mentioned previously, more thorough studies on the XAI, and on relative feature importance, could improve and optimize the model, eliminating superfluous features and strengthening the most important ones.

Bibliography

[1] Roberto Finesso, Ezio Spessa, Mattia Venditti (2014), *Layout design and energetic analysis of a complex diesel parallel hybrid electric vehicle.*

[2] Roberto Finesso (n.d), *Optimization of the control strategy of HEVs.*

[3] Jeremy Jordan (2016), *Normalizing your data (specifically, input and batch normalization),* from https://www.jeremyjordan.me/.

[4] Timo Stöttner (2019), *Why Data should be Normalized before Training a Neural Network,* from https://towardsdatascience.com/.

[5] Md Zahangir Alom et al. (n.d.), *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches.*

[6] Sagar Sharma (2017), *Activation functions in Neural Network,* from https://towardsdatascience.com/.

[7] Vikashraj Luhaniwal (2019), *Forward propagation in neural network – Simplified math and code version*, from https://towardsdatascience.com/.

[8] Mohammed Zeeshan Mulla, *Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?*, from https://towardsdatascience.com/.

[9] Valentina Alto (2019), *Neural Networks: parameters, hyperparameters and optimization strategies*, from https://towardsdatascience.com/.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov (2014), *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.*

[11] Raimi Karim (2018), *Intuition on L1 and L2 Regularization,* from https://towardsdatascience.com/.

[12] James Bergstra, Yoshua Bengio (2012), *Random Search for Hyper-Parameter Optimization.*

[13] Jason Brownlee (2018), *A Gentle Introduction to k-fold Cross-Validation,* from https://machinelearningmastery.com/.

[14] Alejandro Barredo Arrieta et al. (2019), *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI.*

[15] *What is Python? Executive Summary,* from https://www.python.org/.

[16] *Spyder,* from https://docs.anaconda.com/.

[17] Davide Chicco & Giuseppe Jurman (2020), *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,* from https://rdcu.be/cg4EP.

[18] Jason Brownlee (2019), *How to use Learning Curves to Diagnose Machine Learning Model Performance,* from https://machinelearningmastery.com/.

[19] Christoph Molnar (2021), *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable.*

[20] Jason Brownlee (2019), *How to Get Reproducible Results with Keras,* from https://machinelearningmastery.com/.