



**Politecnico  
di Torino**

**DIMEAS**

**Dipartimento di Ingegneria  
Meccanica e Aerospaziale**

**Masters's Degree in Mechanical Engineering  
Laurea Magistrale in Ingegneria Meccanica  
Progettazione Meccanica**

**Machine Learning Methods  
to predict the Fatigue Life of  
Selectively Laser Melted  
Ti6Al4V Components**

**Supervisor:**

**Prof. Davide Salvatore Paolino**

**Candidate:**

**Alessio Centola (289202)**

**Co-Supervisors:**

**Prof. Andrea Tridello**

**Ing. Alberto Ciampaglia**

**Academic year 2022 – 2023**



## Contents

List of Figures .....	5
List of Tables .....	7
1. Abstract and State of the Art.....	8
2. What is Selective Laser Melting (S.L.M.) .....	9
2.1 The challenge of manufacturing Titanium Alloys .....	9
2.2 The advantages of Additive Manufacturing .....	10
2.3 The Selective Laser Melting process (S.L.M.) .....	11
3. Building the Database .....	18
3.1 Fatigue testing.....	18
3.2 The shape of the database .....	20
4. How a neural network works .....	24
4.1 The black Box Model.....	24
4.2 The Structure of the neural network .....	24
4.3 Typical machine learning parameters .....	25
4.4 The activation functions.....	27
5. The Feed-Forward Neural Network (F.F.N.N.).....	30
5.1 Preliminary tuning .....	30
5.2 Preliminary validation.....	31
5.3 Modifying the number of epochs .....	33
5.4 Removing the validation set from the training database.....	34
6. The Physics Informed Neural Network (P.I.N.N.) .....	36
6.1 The correlation matrix.....	36
6.1.1 Laser Power / Laser Scan Speed .....	37
6.1.2 Laser Power · Laser Scan Speed.....	38
6.1.3 Laser Scan Speed / Laser Power .....	38
6.2 Enriching the database of the P.I.N.N.....	39
6.3 Implementing the custom loss function .....	41
6.3.1 How the curves are computed.....	42
6.3.2 How the custom loss function works .....	44
6.3.3 Tuning the r parameter.....	45
6.4 Global evaluation .....	50
7. Bilinear and concavity behavior with the P.I.N.N. ....	56
7.1 changing the structure of the P.I.N.N. ....	56
7.2 Considering the 2 <sup>nd</sup> derivative in the custom loss function .....	59

8. F.F.N.N. with concatenated layers .....	63
9. Pure Bi-Linear Behavior .....	67
9.1 The Bi-Linear Function.....	68
9.2 How the Bi-Linear database is built.....	69
9.2.1 Finding the 4 bilinear parameters .....	70
9.3 Tuning the bi-linear network machine learning parameters .....	72
9.4 Results.....	73
10. Final comparison.....	77
11. Stressing the Neural Networks.....	81
11.1 Inspecting the Removed portions.....	84
11.2 Analyzing the Inner Portions .....	87
12. External Validation .....	91
13. Process Parameter Variation Evaluation.....	97
13.1 Building Orientation .....	98
13.2 Hatch Distance .....	99
13.3 Layer Thickness .....	100
13.4 Speed/Power .....	102
13.5 Conclusions.....	103
14. Defects Characterization Neural Network .....	106
14.1 Neural network tuning and Results.....	109
References.....	112

## List of Figures

Figure 1, Traditional manufacturing via Computerized Numerical Control (C.N.C.) .....	9
Figure 2, Manufacturing the same part via Selective Laser Melting (S.L.M.) .....	10
Figure 3, Finished S.L.M. Parts .....	11
Figure 4, S.L.M. process machine layout .....	12
Figure 5, Scanning Strategies .....	13
Figure 6, S.L.M. process manufacturing detail.....	14
Figure 7, S.L.M. laser operation detail .....	15
Figure 8, Defects types and their correlation to the Scan Speed .....	15
Figure 9, Defects population before and after the H.I.P. treatment .....	16
Figure 10, the H.I.P. (left) and Annealing (right) thermal treatments .....	17
Figure 11, The typical Stress - Life fatigue diagram .....	18
Figure 12, Fatigue loading at $R = -1$ (left) and at $R > 0$ (right) .....	19
Figure 13, Rotating bending (left) and tensile (right) fatigue testing apparatus.....	19
Figure 14, The black box model .....	24
Figure 15, The structure of a simple neural network.....	25
Figure 16, Loss behaviour as a function of the number of epochs depending on the learning rate.....	26
Figure 17, Linear activation function.....	27
Figure 18, Sigmoid activation function .....	27
Figure 19, Rectified Linear Unit (ReLU) activation function .....	28
Figure 20, The hyperbolic tangent (Tanh) activation function .....	28
Figure 21, M.S.E. as a function of the number of epochs.....	31
Figure 22, Gong2015 SLM -MP 4 dataset.....	32
Figure 23, Gong2015 SLM-MP 4 Preliminary result .....	33
Figure 24, Increasing the number of epochs results in an overfitting behaviour.....	34
Figure 25, Typical Bi-Linear behaviour of fatigue curves .....	34
Figure 26, Preliminary fatigue curve after removing the dataset from the training database..	35
Figure 27, Correlation matrix of the training database .....	36
Figure 28, Correlation matrix with Power/Speed feature cross .....	37
Figure 29, Correlation matrix with Power * Speed feature cross .....	38
Figure 30, Correlation matrix with Speed/Power feature cross .....	38
Figure 31, Fatigue curve after enriching the database .....	40
Figure 32, Fatigue curve after using the Tanh activation function .....	41
Figure 33, The custom penalization function .....	43
Figure 34, Change of behaviour with respect to the coefficients a and b .....	45
Figure 35, R.M.S.E.(Nf) as a function of r for the 3 analysed datasets .....	47
Figure 36, Sweet spots for the 3 datasets .....	47
Figure 37, Fatigue curves corresponding to their sweet spots .....	48
Figure 38, Final sweet spot choice.....	49
Figure 39, Fatigue curves corresponding the final choice of the r parameter.....	49
Figure 40, Fatigue curves after the tuning of the r parameter without removal from the database.....	52
Figure 41, R.M.S.E.(Nf) evaluated for all the datasets.....	53
Figure 42, Datasets evaluated with the tuned Physics Informed Neural Network .....	55
Figure 43, Typical fatigue curve .....	56

Figure 44, Activation functions used for the 3rd type of Neural Network .....	56
Figure 45, Evaluation of the 3 usual datasets with the 3rd Neural Network before implementing the 2 <sup>nd</sup> derivative.....	58
Figure 46, Fatigue curves with the concavity facing upwards with the 2 <sup>nd</sup> and 3 <sup>rd</sup> Neural Network type.....	58
Figure 47, customized penalization function to take into account the 1st and 2nd derivative	59
Figure 48, Improvement of the concavity behaviour for the set DuQian2020, 7 .....	60
Figure 49, Improvement of the concavity behaviour for the set Alegre2022, As Built .....	61
Figure 50, Curves, that after penalizing negative 2 <sup>nd</sup> derivatives, still have the concavity facing downwards .....	62
Figure 51, Logical scheme for the concatenated feed forward neural network.....	63
Figure 52, Three usual datasets evaluated with the 4th neural network type: Concatenated F.F.N.N. ....	64
Figure 53, Datasets evaluated with the 4th type of neural network: concatenated F.F.N.N....	66
Figure 54, History of the 4 neural network types seen up to now .....	67
Figure 55, Mathematical layout for the Bi-Linear approach .....	68
Figure 56, Logical layout to find the best combination of the 4 Bi-Linear parameters.....	71
Figure 57, Usual three datasets evaluated with the 5th method: Bi-Linear F.F.N.N.....	74
Figure 58, Datasets evaluated with the 5th type of Neural network: Bi-Linear F.F.N.N. ....	76
Figure 59, R.M.S.E.(Nf) for all the datasets for the 5 types of neural networks .....	77
Figure 60, Comparison of the 5 methods for the 3 usual datasets .....	78
Figure 61, Datasets evaluated with all the 5 methods at the same time for comparison .....	80
Figure 62, All the fatigue points in the S - N diagram.....	81
Figure 63, Training database with the points to be removed highlighted in red .....	83
Figure 64, Trimmed training database used to stress the neural network.....	83
Figure 65, Right bottom region stressed network analysis .....	84
Figure 66, Left bottom region stressed network analysis .....	85
Figure 67, Left top region stressed network analysis .....	86
Figure 68, Right top region stressed network analysis .....	87
Figure 69, Usual three datasets analysed in the stressed condition .....	88
Figure 70, Datasets in the inner region evaluated in the stressed conditions .....	90
Figure 71, The 3 datasets of the article Xu2020 .....	93
Figure 72, The 3 datasets of the article Xu2020 evaluated.....	95
Figure 73, Varying the Building Orientation.....	99
Figure 74, Varying the hatch distance .....	100
Figure 75, Varying the Layer Thickness.....	101
Figure 76, Varying the Speed/Power cross-parameter .....	102
Figure 77, Best combination of process parameters .....	103
Figure 78, Best combination of process parameters with thermal treatments comparison ...	104
Figure 79, Gumbel Plot example .....	106
Figure 80, Evaluation of all the Gumbel plots.....	111

## **List of Tables**

Table 1, The shape of the training database.....	22
Table 2, preliminary Hyper-parameters .....	31
Table 3, Gong2015 SLM-MP 4 .....	32
Table 4, Neural network preliminary evaluation dataset .....	32
Table 5, Feature cross attempts and their correlation coefficients.....	38
Table 6, Updated training database with feature cross .....	39
Table 7, Updated Neural network evaluation database .....	42
Table 8, All the possible values of the r parameter.....	45
Table 9, List of all the datasets .....	51
Table 10, 3rd Neural Network structure .....	57
Table 11, Training database for the Bi-Linear approach .....	70
Table 12, Structure and Hyper-parameters of the Bi-Linear F.F.N.N. ....	72
Table 13, Strategy used to stress the Neural Networks .....	82
Table 14, Features for the external validation article .....	91
Table 15, Experimental points for the 3 datasets of the article Xu2020.....	92
Table 16, Process Parameter variation evaluation database .....	98
Table 17, Example procedure to obtain a Gumbel plot .....	107
Table 18, Defects Neural Network training database .....	108

## 1. Abstract and State of the Art

The aim of the proposed methodology is to define a relation between the process parameters and the fatigue life of Ti6Al4V components, produced via Selective Laser Melting technologies. The method presented in this thesis adopts Machine Learning techniques and their several types of Neural Networks, as an attempt to reduce the cost of further fatigue testing, as well as to predict the life of components, in relation to the process parameters, thermal treatments and surface treatments. As it is well known, fatigue testing is rather expensive and time consuming. The implementation of machine learning and the extrapolation of fatigue data on the fatigue properties of titanium specimens is what helps in finding a correlation between the several process parameters, the thermal and surface treatments performed postproduction and the fatigue life. The idea is to get a sort of “program” that, after having inserted all the aforementioned parameters, outputs a fatigue curve that can be used to have an idea on the life of a components subjected to a determined stress amplitude level, without any time and money expenditure.

When it comes to the literature, this type of work is relatively new. Antriksh Sharma [1], from the Arizona State University, made a thesis at the end of July 2020, in which he had developed a data driven approach to predict the static and fatigue properties of additively manufactured Ti-6Al-4V. He analysed several additive manufacturing technologies, like S.L.M., E.B.M. and D.M.L.S. to create two interconnected neural networks. The first one, from the process parameters only (no surface treatments have been considered) is capable of finding the tensile properties and from those ones the second neural network predicts the two coefficients of the fatigue curve. Liu and Chen [2] wrote an article by the end of January 2021, in which they have used another training approach with probabilistic guided learning, via a smaller database and a Probabilistic Physics Informed Neural Network (PPINN). Lastly, Hornas [3] published his work at the end of December 2022, (in the same period on which the following work was under development) in which he fatigue tested a couple of dozens of specimens and he established a database through the fatigue life and the stress amplitude of those samples and in parallel he characterized, via micro CT scans, the overall population of defects in terms of size, location and shape. However, the first two approaches have sets with missing process parameters. The work described in the next pages, has a fully populated training database with a higher numerosity, while having a more direct and straight forward training approach. Several other peculiarities will be discussed in the following chapters.

## 2. What is Selective Laser Melting (S.L.M.)

When it comes to manufacturing mechanical parts, throughout the years, it has been shown that traditional manufacturing methods work nicely when we have to produce shapes that are axisymmetric and semiregular. The most complex shapes that we can think of are turbine blades or compressor impellers. Those ones can be reproduced without any significant difficulty by the most advanced C.N.C. machines. However, we must remember that they are made of steel, that, even if it has a composition that makes the alloy hard, it is still easier to be manufactured with respect to titanium alloys.



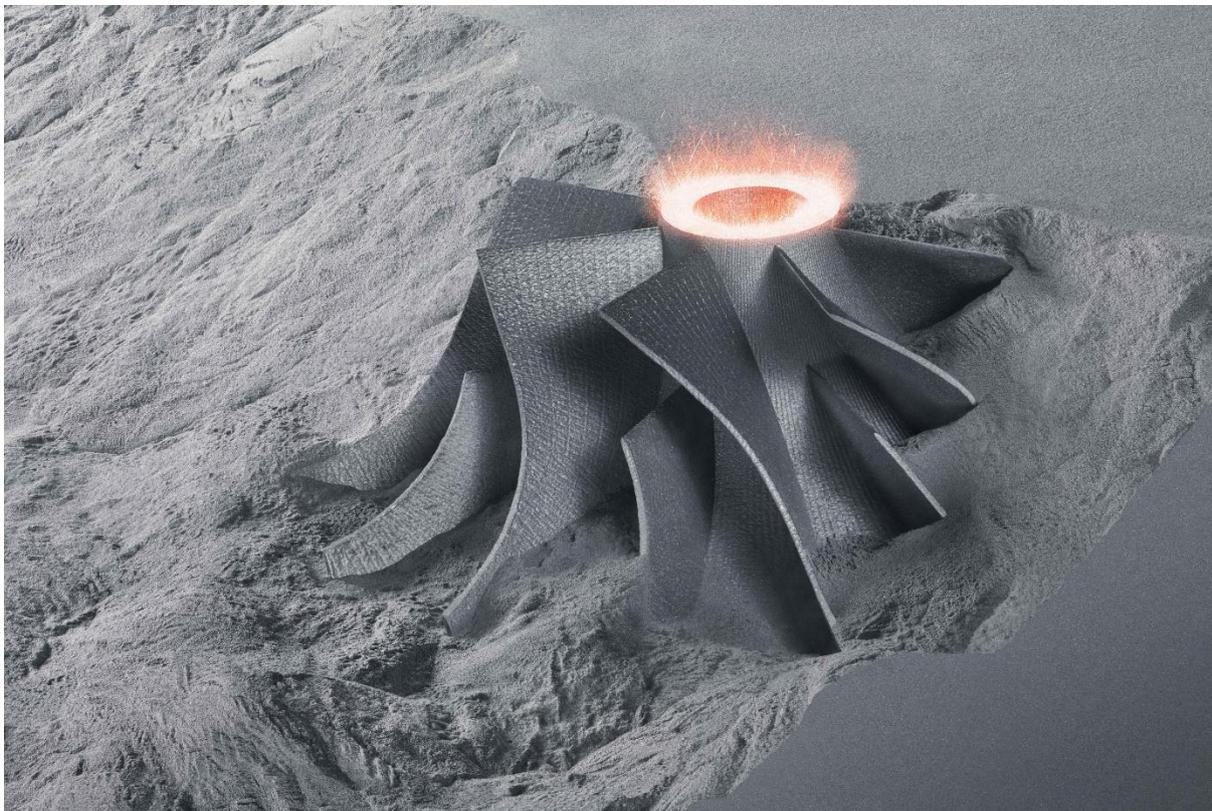
*Figure 1, Traditional manufacturing via Computerized Numerical Control (C.N.C.)*

### 2.1 The challenge of manufacturing Titanium Alloys

Titanium alloys are much harder to shape with traditional methods. In fact, the characteristic that makes titanium difficult to design and machine is, at the same time, one of its best characteristics. Titanium is not capable of conducting heat efficiently. This property is fantastic for parts that have to work at high temperatures, like the previously mentioned turbine blades. But this, at the same time, means that the cutting tool will absorb nearly all the heat that is generated during manufacturing, causing it to be degraded sooner. Not only, but titanium is also tough, and a high shear force is needed to create a chip. As if that wasn't enough, Titanium, even though it's not so strong at room temperature, is one of

the strongest metals when heated up to the point of glowing red, because it doesn't lose much strength when heated up with a thousand degrees. These reasons make machining titanium tedious and more expensive.

When producing typical shapes someone may argue that using additive manufacturing is an exaggerated choice, but if we consider that titanium alloys are often employed for biomedical applications, whose parts and implants are of complicated and often personalized shapes, this technology is an obvious choice. In this work it has been chosen to analyze, among the various technologies available (electron beam melting E.B.M., laser power bed fusion L.P.B.F., direct energy deposition D.E.D. and selective laser melting S.L.M.) Selective Laser Melting, as it is one of the most popular and used methods.



*Figure 2, Manufacturing the same part via Selective Laser Melting (S.L.M.)*

## **2.2 The advantages of Additive Manufacturing**

With respect to the traditional subtractive manufacturing techniques, in which material is subtracted from a casted block, additive manufacturing is based, as the word itself explains, on adding material layer by layer. Each layer is connected via thermal bonding to the next and the previous one, up until the desired shape is reached. Additive manufacturing employs computer aided design technologies (C.A.D.) that are used by the S.L.M. machine to selectively melt and bond the powder. Still, thermal and finishing processes are used, to improve the

mechanical properties of the component as well as to achieve the correct sizes and tolerances. In fact, the tensile properties of S.L.M. as built parts (as built means as produced out of the S.L.M. machine, without any thermal or finishing process) are comparable to traditionally produced parts. While, in turn, the fatigue properties are worse with respect to conventionally produced components. This justifies the employment of thermal and surface treatments. The most used thermal treatments on titanium alloys are annealing and hot isostatic pressing (H.I.P.). When it comes to surface treatments, we have a wide selection: surface machining, sand blasting, shot peening, surface mechanical attrition treatment (S.M.A.T.), laser shot peening and polishing. Something else to be considered, is the fact that the machine that is printing the component layer by layer, can operate without having a human to perform intermediate operations. To be conservative, it's just needed a supervisor. In addition, when it comes to the complexity of the shape to be manufactured, this does not affect the cost of the production as it would happen in traditional manufacturing techniques. In fact, additive manufacturing is highly flexible and customizable.



*Figure 3, Finished S.L.M. Parts*

### **2.3 The Selective Laser Melting process (S.L.M.)**

The selective laser melting process can be thought of as a sort of evolution of the selective laser sintering process. It is capable of delivering complex shapes with properties that are comparable to the ones of the traditionally manufactured components, when it comes to tensile properties. However, as it was anticipated before, we need to perform additional thermal and surface treatments if we want

to obtain a fatigue behavior that is comparable to the one of traditionally produced parts.

In the chamber of an S.L.M. machine we have a bed of metal powders that are ready to be hit by a laser. The *Powder Size* [ $\mu\text{m}$ ] is usually described by a gaussian distribution that is provided by the manufacturer, but it is also common to find just the *Powder Size Maximum* [ $\mu\text{m}$ ] and the *Powder Size Minimum* [ $\mu\text{m}$ ], or directly its *mean value*. This laser then hits the powders, melting them and solidifying gradually the first layer, as it is dictated by the C.A.D. file of the part to be manufactured. Once the first layer is completed, a blade passes over another container that is stocking the powder that is going to be processed in the next steps. But before doing this, the piston that is beneath the stocking pool is raised by the same distance that the piston below the building chamber will travel downward. This distance is called *Layer Thickness* and it's usually measured in  $\mu\text{m}$ . The blade then recoats a layer of powder, that is as thick as the parameter layer thickness itself. The laser passes again, completing another layer and the process is repeated several times, until the part is completed.

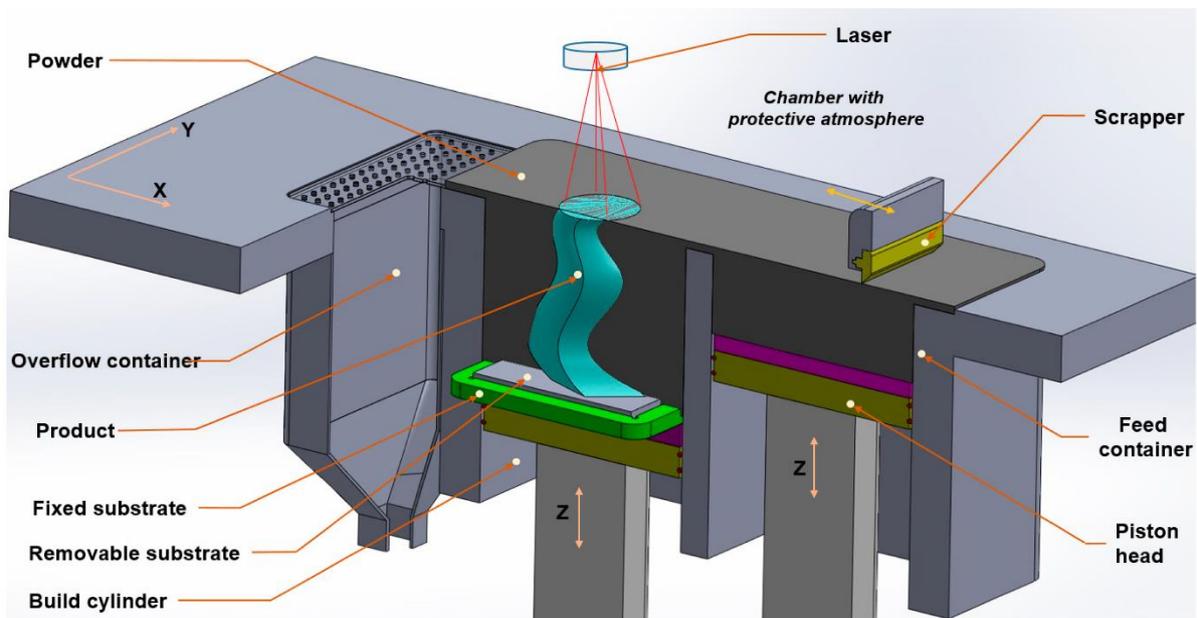


Figure 4, S.L.M. process machine layout

This occurs in a chamber that has a protected atmosphere with inert gases, to avoid any type of unwanted chemical bond. Moreover, the manufacturer has the freedom to decide in which direction the part can be built. In the following work, 3 main configurations that identify the *Building Orientation* have been observed: horizontal [ $0^\circ$ ], vertical [ $90^\circ$ ] and inclined [ $45^\circ$ ].

Moving to the laser, since it has to cover an area that is wider than its *Laser Spot Size* [ $\mu\text{m}$ ], a *Scanning Strategy* is needed. As *Figure 5* shows, we can choose between several strategies: Monodirectional parallel, Bidirectional parallel, Contouring, Island Strategy... and in addition, each layer can have its own scanning strategy. The possibilities are practically endless, allowing the manufacturer to ensure optimal bonding between the subsequent layers.

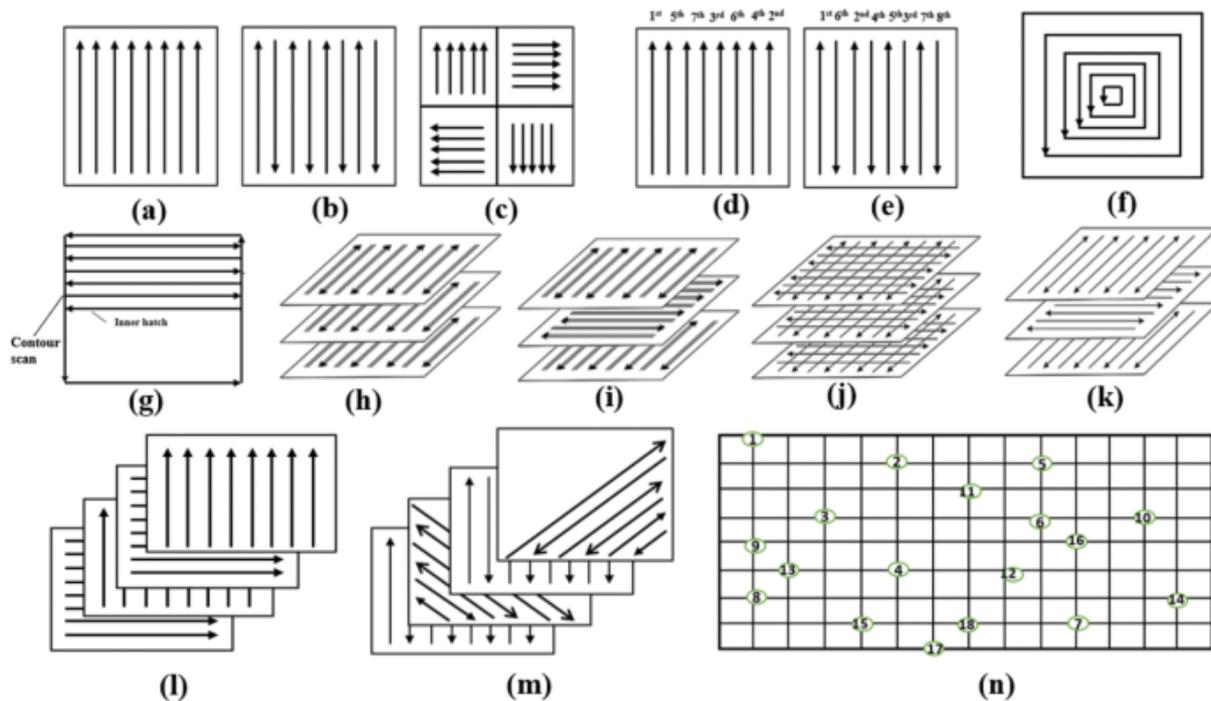


Figure 5, Scanning Strategies

But, even though we can be creative with the strategy, it is useless to do so if we don't have enough space between one pass and the other. Therefore, another important parameter that has to be carefully chosen is the *Hatch Distance* [ $\mu\text{m}$ ], that is the distance between one laser pass and the next one. Too much hatch distance means that we can have unmelted powder between two subsequent passages. Too little means having long time to produce the part as well as having multiple remelting of the same section of the layer, which can be optimal in several cases, but also detrimental in other; it all depends on how all the process parameters are set together.

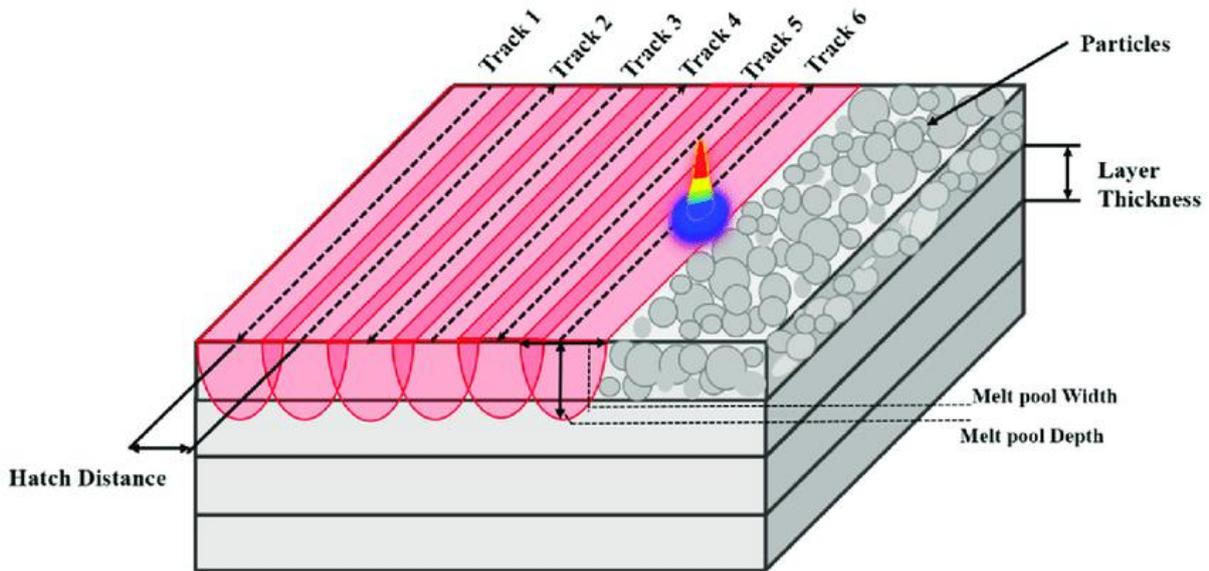


Figure 6, S.L.M. process manufacturing detail

When hitting the powders, the S.L.M. machine has also the option to choose the **Laser Power [W]** and the **Laser Scan Speed [mm/s]**. The first one dictates how much power the laser is delivering to the powders, while the latter one will determine how quickly the laser will pass to create the layer of material. If the speed is too high and the power is too small, we may not have enough time to melt the powder. Vice versa we can have so much power and time to melt the powder, that the multiple layers that were previously formed can be remelted again and again, ruining the quality of the finished part. To ensure that the part has an optimal bonding with the building platform, it is a good practice to heat it up. Usually, the **Plate Temperature** is set to 200 °C for Ti6-Al4-V titanium alloys.

Another important parameter to keep in mind is the **Energy density [J/mm<sup>3</sup>]**. It greatly affects the properties of the part and it's defined by the equation below:

$$E = \frac{P}{v \cdot h \cdot t}$$

In the previous equation **P** is the **laser power [W]**, **v** is the **scan speed [mm/s]**, **h** is the **hatch distance [mm]** and **t** is the **layer thickness [mm]**. The Energy density allows us to understand the energy that is involved in the volume in which the laser is operating and it's also strictly related to the porosity of the finished part. Similarly, to the hatch distance, the **Point Distance [μm]** is a key indicator that explains how each spot (since the laser hits the surface intermittently at a high frequency while moving) in which the laser, is operating is distant between the previous spot. Again, if it is too high, we could have unmelted regions between two subsequent spots; while if it is too small, multiple remelting of the same section of the spot can occur, degrading the quality of the product.

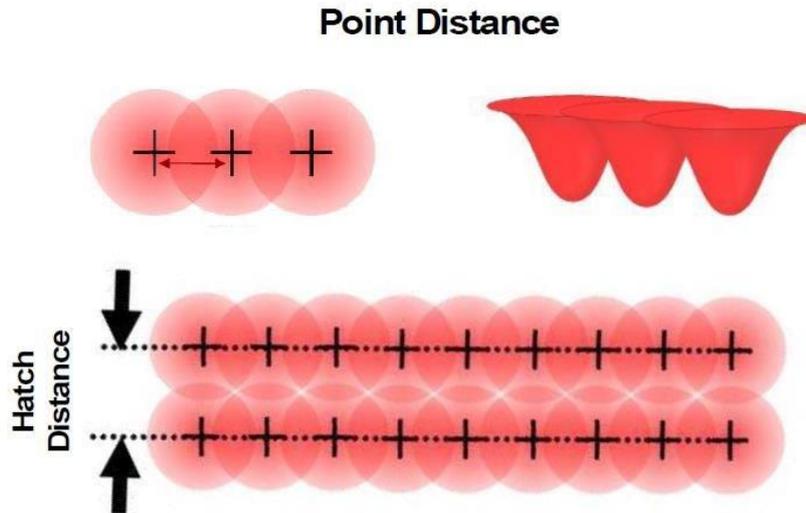


Figure 7, S.L.M. laser operation detail

When manufacturing *Exposure Time* [ $\mu\text{s}$ ] must also be monitored accordingly, because it has a significant impact on the surface energy, which increases if the exposure time itself is increased. By varying the previously mentioned process parameters, different mechanical properties can be obtained, therefore optimal combinations of those, have to be carefully chosen to ensure the best properties possible. Another aspect that has to be considered is linked to the density of the produced part, in relation to the density of its traditionally produced counterpart. In fact, even if we are aiming to get a 100% density, we might end up getting values around the 98% of the density of its bulk traditional part. This occurs because some gas is locked in the material during fusion, or because the powder particles didn't have an optimal position while melting. This can cause one of the most feared defects of S.L.M.: the *Lack of Fusion* (L.O.F., Figure 8). This kind of defect is one of the main causes of the poor fatigue life of as built parts.

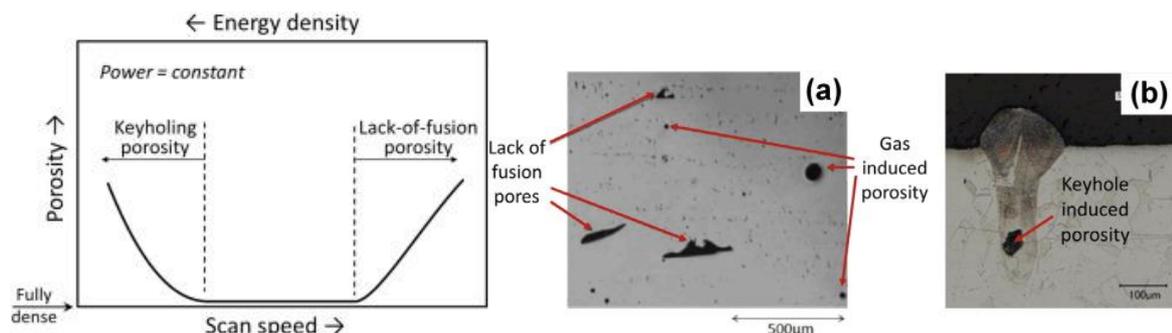


Figure 8, Defects types and their correlation to the Scan Speed

For example, if we have a scan speed that is too high, the powder doesn't have enough time to be completely melted, leaving the microscopical sphere (grain of powder) partially unmelted. At the same time, if the scan speed is too low, we

can have keyhole induced porosities. This phenomenon is great evidence on how crucial the parameter tuning is.

To solve these kinds of problems, coming from the porosities and defects, the **Hot Isostatic Pressing** thermal treatment (H.I.P.) can be employed. In fact, the Hot Isostatic Pressing is a process that has been already used frequently in traditionally manufactured titanium castings, able to reduce the internal porosities and therefore capable of increasing the fatigue life in a remarkable manner. When it comes to HIPing, high pressures (1000 to 2000 bar) and high temperatures (800 to 1050 °C) are simultaneously employed for a couple of hours (2 to 3 hours), in a specific chamber filled with inert gas, to the part itself.

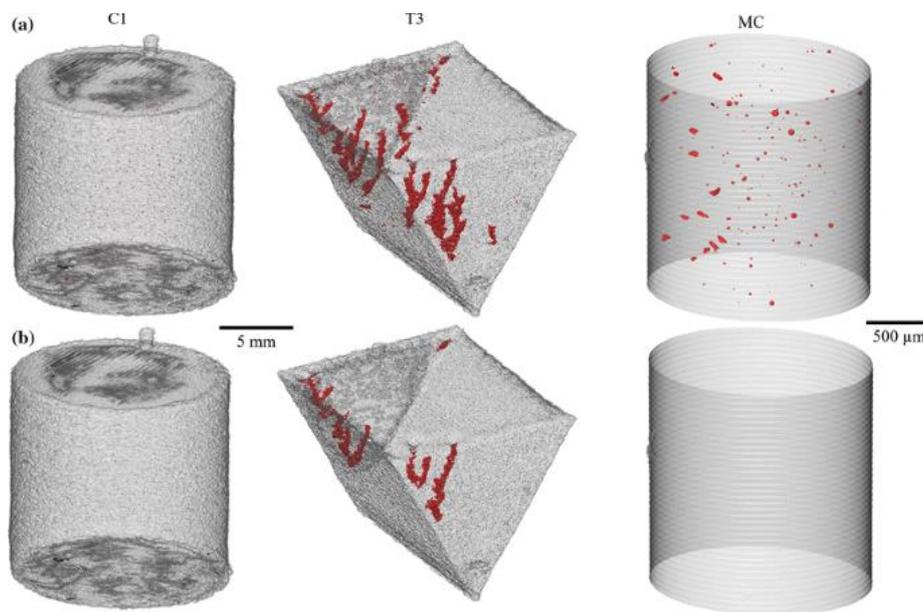
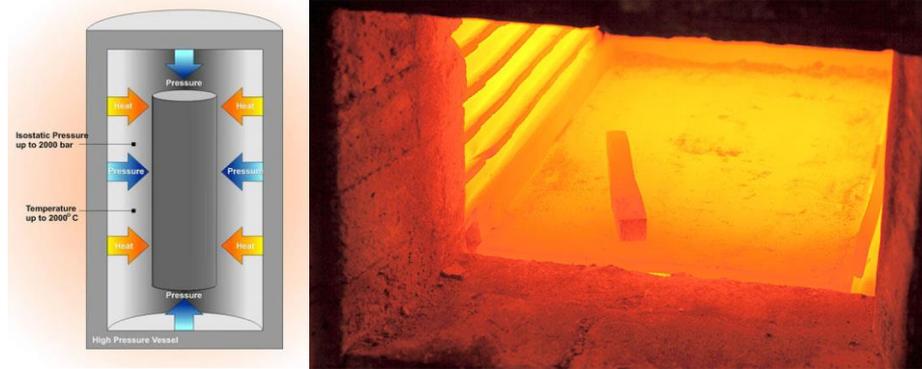


Figure 9, Defects population before and after the H.I.P. treatment

In addition, as fabricated parts do not have a nice surface finish and, as it is well known, bad surface quality results in poor fatigue life. To enhance the surface quality, we can use one or more of the several surface treatments available like **laser shot peening**, **shot peening**, **polishing**, **surface mechanical attrition treatment** (S.M.A.T.), **surface machining** and **sand blasting**. Since we are working at high temperatures with the powders and since their internal temperature varies from low values to high values quickly, the final part can undergo distortions and residual stresses. The gradual solidification of each layer creates high thermal stresses that are responsible of ruining both the mechanical properties and the shape of the component. To solve the residual stress problem, we can employ the **Annealing** thermal treatment, which involves reheating the component in a furnace at a temperature (500 to 950 °C) significantly below the melting point (1660 °C) for enough time (30 minutes to 4 hours).



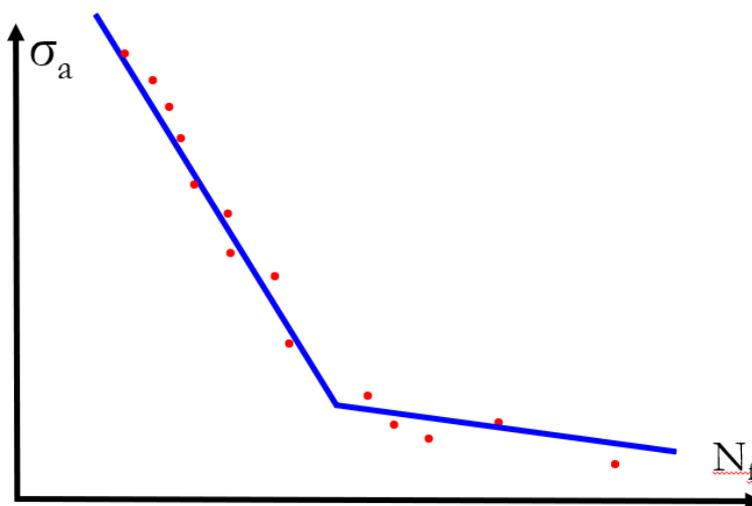
*Figure 10, the H.I.P. (left) and Annealing (right) thermal treatments*

### 3. Building the Database

When it comes to applying machine learning techniques, a database to store information to train the neural network is needed. Generally speaking, there are few rules to be followed in order to have a neural network that is as precise as possible: collecting as much data as possible, having as many features as possible (the features are the columns that are needed to predict the data that we are interested in) and, at the same time, ensuring that the data is coming from a trustful source. In this case, the interest was to predict the titanium alloy Ti6-Al4-V fatigue curves starting from the process parameters of the selective laser melting process. The data was collected from articles coming from the Elsevier Science Direct article search engine Scopus, in which the researchers conducted studies on the fatigue behavior of specimens that were manufactured by S.L.M.. Nearly 100 articles have been read carefully, but of those only the one reporting the process parameters, the information on the thermal and surface treatments and the fatigue curves, have been chosen to extrapolate data from. Since not always, the information on the fatigue life was reported numerically, the software Engauge Digitizer was used to manually extract the points from the SN diagram. While extracting the points to then make predictions about the fatigue life, further numerical data concerning the defect characterization of broken specimen, was carefully searched, still making sure to be able to connect those numbers to the process parameter. This part will be explained in another section.

#### 3.1 Fatigue testing

When it comes to fatigue, the typical SN diagram is made in this way (*Figure 11*):



*Figure 11, The typical Stress - Life fatigue diagram*

On the X axis we have the number of cycles to failure  $N_f$  [Cycles], while on the Y axis we have the stress amplitude  $\sigma_a$  [MPa]. Fatigue testing is characterized by specimens that are tested in special machines with usually sinusoidal loads.

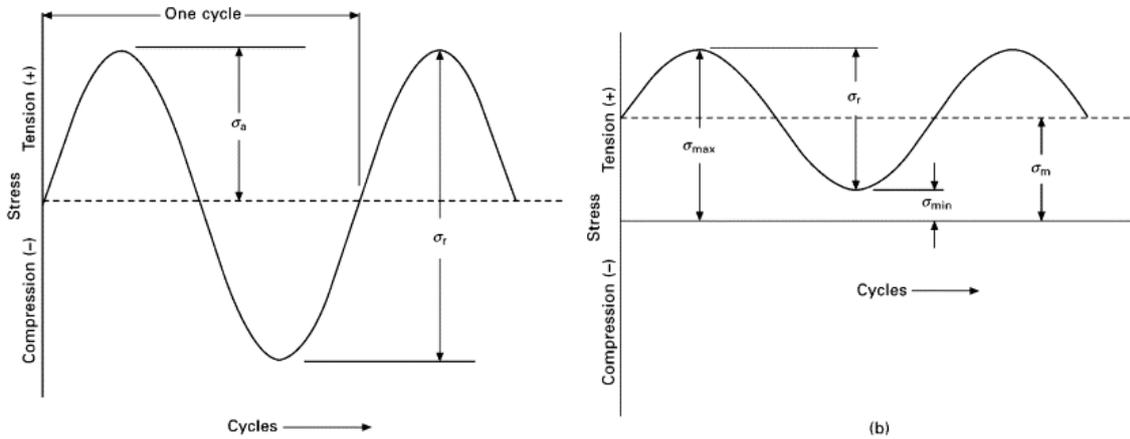


Figure 12, Fatigue loading at  $R = -1$  (left) and at  $R > 0$  (right)

We can have the typical rotating bending configuration testing (Figure 12 and 13, left), in which the specimen is mounted in a machine that creates a bending moment inside the material while it is rotating, thus allowing it to be cyclically loaded, or the uniaxial loading testing (Figure 12 and 13, right), where the specimen is mounted in a tensile testing machine, that elongates the part up to a maximum value, to return then to a smaller force of tensile loading.

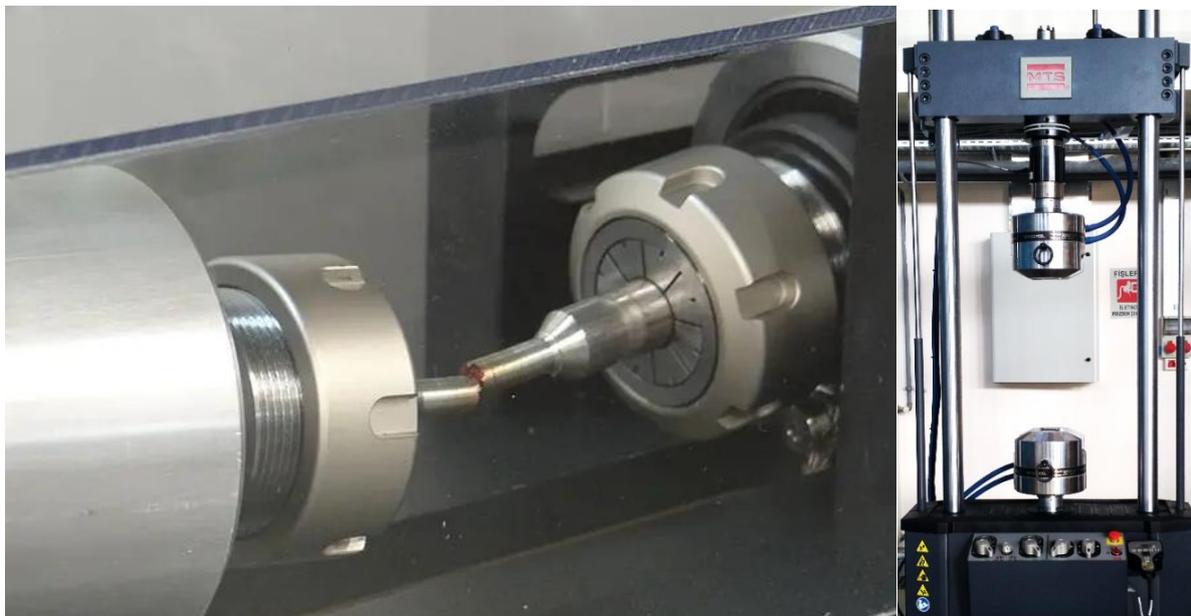


Figure 13, Rotating bending (left) and tensile (right) fatigue testing apparatus

The specimens undergo several cycles at a level of stress amplitude  $\sigma_a$ , until the failure is reached. When this happens the number of cycles to failure for the specimen is registered as  $N_f$ , thus creating one of the red points in the SN diagram. One of the most important parameters to keep in mind while performing fatigue testing is the *stress ratio R*.

$$R = \frac{\sigma_{min}}{\sigma_{max}}$$

This latter one basically explains how the sinusoidal of the stress-time diagram is displaced with respect to the X axis on *Figure 12*. Given  $\sigma_{max}$  as the maximum of the sinusoidal stress-time diagram and  $\sigma_{min}$  as the minimum, the stress amplitude  $\sigma_a$  is calculated as:

$$\sigma_a = \frac{\sigma_{max} - \sigma_{min}}{2}$$

While, on the other hand the *mean stress* value  $\sigma_m$  is obtained in this way:

$$\sigma_m = \frac{\sigma_{max} + \sigma_{min}}{2}$$

Referring to *Figure 12*, it is evident that the mean stress allows us to understand how the sinusoidal is placed with respect to the X axis. In fact, for the rotating bending case ( $R = -1$ ), since  $\sigma_{max} = -\sigma_{min}$ , the mean value  $\sigma_m$  is null, while the alternate stress value  $\sigma_a = |\sigma_{max}| = |\sigma_{min}|$ . For the uniaxial tensile loading the situation differs, and it depends on how the testing parameters have been set up. For sure  $R \neq -1$ , and both  $\sigma_{max}$  and  $\sigma_{min}$  are positive, meaning that the entire curve is above the X axis and that  $\sigma_m$  is positive. Then the value of  $R$ , which is typically **0.1** or **0.2**, depends on how distant  $\sigma_{max}$  and  $\sigma_{min}$  are between each other.

The traditional fatigue testing is performed under the rotating bending condition. However, several researchers have used the uniaxial tensile loading. Therefore, to grant continuity between the two methods, the Smith-Watson-Topper correction (**S.W.T.**) was used to bring the fatigue points that have been tested in the uniaxial tensile loading ( $R \neq -1$ ) to the rotating bending condition ( $R = -1$ ). Once several specimens have been tested at several levels of stress amplitude, the fatigue curve can be produced. Usually, the curve is a simple line or a bi-linear model, that can be used to have an idea of how long a component can survive if it is loaded at a stress amplitude level. The aim of this work is to predict those fatigue curves from the process parameters of the specimen.

### 3.2 The shape of the database

The database is constructed to be fed inside the Python programming environment and to be read, as a CSV file, by the Python library named “*pandas*”.

Therefore, it will have as many rows as the number of fatigue points that have been digitized from the articles, and as many columns as the features that are used to predict the fatigue life. In machine learning literature, the variables that are used to predict what we are interested in are called *features*, while the variables to be predicted are named *labels*. It is of extreme importance to make sure that we do not have blank spots in the database, otherwise the machine learning algorithm won't work. In this case, while extracting data from the articles, it has been found that not all the articles are continuous between each other, in the sense that some of them are reporting some process parameters, while others are missing some of them. In order to fulfill the previous statement, all the articles that were not reporting the fundamental process parameters (*orientation, scan speed, power, hatch distance and layer thickness*), were discarded. Unluckily, from all those articles, just a few had information regarding the *powder size, spot size, scanning strategy, plate temperature, point distance and exposure time*. These latter ones were not used to predict the life.

Then, inside the articles, information concerning the surface and thermal treatment was found in several testing sets. In this case, when one of these treatments was performed, a boolean value was used to note if the treatment was implemented (number 1) or not (number 0). For the thermal treatments, as it was anticipated before, *annealing* and *HIP* were noted, as well as the *annealing temperature* (if the annealing was not done, the annealing temperature was reported as 20 °C). While for the surface treatments, again a boolean strategy was chosen to denote whether the treatment has been performed or not. Those treatments are *machining, sandblasting (S.B.), shot peening (S.P.), laser shot peening (L.S.P.), surface mechanical attrition treatment (S.M.A.T.), electric discharge machining (E.D.M.) and surface polishing*.

Then, when digitizing the points on the fatigue diagram, the stress amplitude  $\sigma_a$  and the life  $N_f$  of the specimen was carefully noted. In this case the stress amplitude  $\sigma_a$  was used as a feature to predict the label fatigue life  $N_f$ . This was done to replicate what happens in fatigue testing: usually in the machine is set the stress amplitude level  $\sigma_a$  that the specimen will undergo and, as a result, we get the number of cycles  $N_f$ . The opposite strategy could have been used anyways and it is assumable that the results shouldn't have varied that much. Another important note is the fact that it has been decided to discard runouts, because they need to be treated in a specific manner that is outside the scope of this work.

To keep the workflow organized, the lines that were belonging to the same article have been labeled by an additional column called *Article*. While below the column *Name*, it was reported the name of the subset to which the dataset was belonging, concerning the same article. For example, there are articles that tested as built specimens, annealed specimens and HIPed specimens; in this case below

the column “*article*”, the same name was reported, but below the column “*name*”, three names have been used to distinguish between those three treatments.

Lastly below the column *Code*, a number ranging from 0 to the number of subsets (76 in total, so 75) has been used. This has been useful to work smoothly inside the programming environment. To work with the *pandas* library, it is of uttermost importance to have the name of the column (without spaces) reported.

The following figure illustrates how the database is built:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1																				
2																				
...																				
i																				
...																				
N																				

Where:

<b>A</b>	Article Subset Name	<b>K</b>	H.I.P. Boolean
<b>B</b>	Article Name	<b>L</b>	Machined Boolean
<b>C</b>	Subset Global Code	<b>M</b>	Sand Blasted Boolean
<b>D</b>	Build Orientation [°]	<b>N</b>	E.D.M. Boolean
<b>E</b>	Laser Power [W]	<b>O</b>	L.S.P. Boolean
<b>F</b>	Laser Scan Speed [mm/s]	<b>P</b>	S.P. Boolean
<b>G</b>	Hatch Distance [mm]	<b>Q</b>	S.M.A.T. Boolean
<b>H</b>	Layer Thickness [µm]	<b>R</b>	Surface Polishing Boolean
<b>I</b>	Annealed Boolean	<b>S</b>	Stress Amplitude (at R= -1) [MPa]
<b>J</b>	Annealing Temperature [°C]	<b>T</b>	Fatigue Life $N_f$ [Cycles]

Table 1, The shape of the training database

Of course, the columns in yellow noted by the letters A, B and C, are not useful to train the neural network, therefore, before the training phase, they have to be removed. Their usage is limited only to recognize the subset and to give it a title and a code while working in python.

Usually in machine learning techniques it is useful to start predicting the label with a few features and then using more of them to improve the precision of the predictions. In the first attempts, only the process parameters (D, E, F, G and H in green) and the stress amplitude (S), have been used. While going forward, more and more treatments were considered. It's also useful to produce the so-called feature crosses, in which two or more features are combined by multiplication or division. In a later part of the work, it was observed that producing a feature cross

between the laser power and the laser scan speed improved significantly the quality of the predictions. This idea originated from the relation that lies between those two parameters. In fact, as it was stated in *Section 2.3*, it is needed an optimal combination of the two. In addition, by studying the correlation matrix of the training database, it was observed that using the feature cross Speed/Power was giving a correlation coefficient (with respect to  $N_f$ ), that was higher than the correlation coefficients of the speed and the power alone, thus improving the predictions.

## 4. How a neural network works

Since performing fatigue testing is rather lengthy and expensive and, at the same time, the production of the specimens, with all the thermal and surface treatments is surely not cheap, machine learning is undoubtedly a viable choice.

### 4.1 The black Box Model

We can think of a machine learning problem as a *neural network* (N.N.) that, at the beginning, is comparable to a black box model. We have the inputs, that are the features ( $x_i$ ), that are used to predict the outputs, the so-called labels ( $y_i$ ).

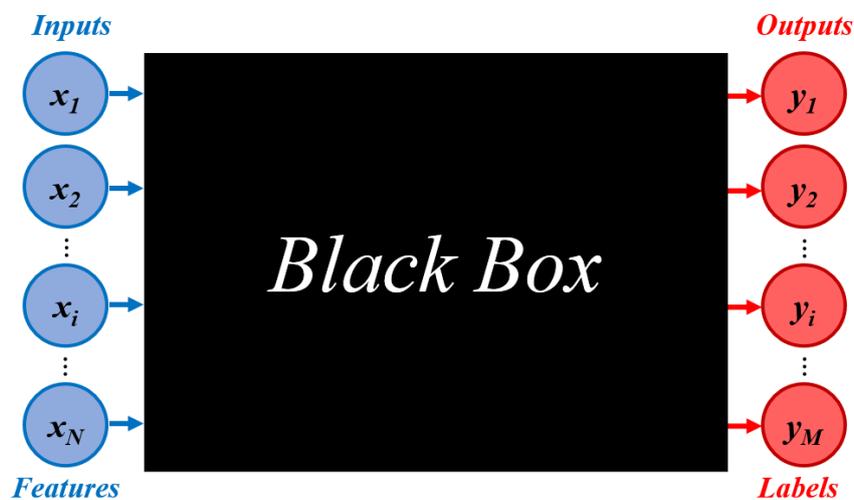


Figure 14, The black box model

But, before using this model to predict what we are interested in, it is needed to first train the model in a manner that produces satisfactory results. To do this we have to feed the code with the training database, that is the one that has been explained in *Section 3*. In this way the neural network is learning how to correlate the inputs to the outputs to produce a prediction.

But why is it called neural network? If we think of a machine learning problem as a black box, we cannot answer this question; but, on the other hand, if we understand that inside this box there is an intricate series of connections, we can easily understand why it is called in such way.

### 4.2 The Structure of the neural network

In fact, a neural network can be thought of way to process data in a manner that is inspired to the human brain, that's why it is also called *artificial intelligence*. Inside this so-called black box, we have several nodes that are interconnected between each other. Since this kind of structure, resembles the way the human brain is built, those nodes are called *neurons* and therefore, the reason why this is called neural network it's trivial.

A point that comes to our advantage, is that we are able to specify how those neurons are linked between each other and how many of them are present.

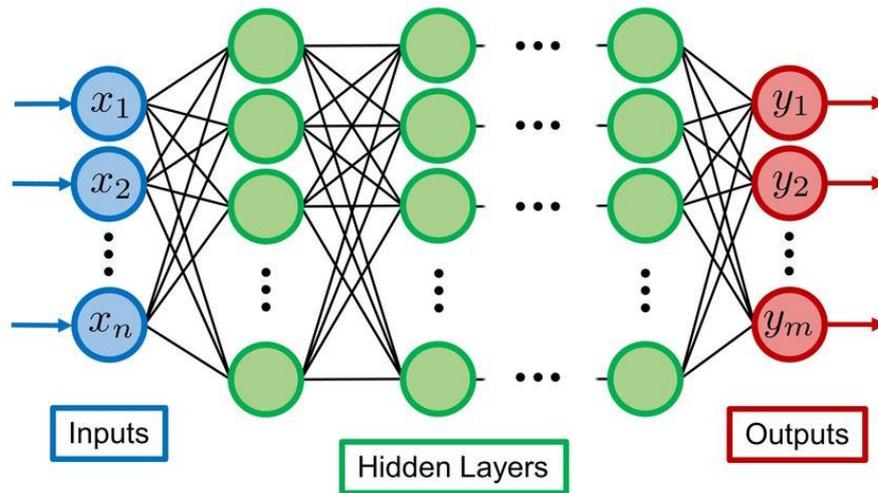


Figure 15, The structure of a simple neural network

We can input how many hidden layers we have and how many neurons per layer are present.

The input layer takes the signal that is then moved to the next hidden layer, while this one takes it to the next hidden layer and so on up until the output layer, from which we are getting the final result.

But what is really going on inside the hidden layers? Those ones are also known as dense layers because they are having way more nodes than the input and the output layers. As it was stated before those nodes are called neurons and they are units that work together by taking the signal from the input and producing a *weight* ( $w$ ) and the *bias* ( $b$ ) that are combined with the input features ( $x$ ) to produce a *sum* ( $g$ ) that is calculated in such way:

$$g_i = w_i \cdot x_i + b_i$$

### 4.3 Typical machine learning parameters

Then this sum  $g$  is fed through a particular function that has to be specified per each layer called **activation function** ( $f_a$ ), which is then sent to the next layers up until the output one:

$$y_i = f_{a,i}(g_i)$$

Another important parameter to be chosen, while performing machine learning tasks, is the **number of epochs**. The process that has been stated before is repeated as many times as the number of epochs. Therefore, it follows that, as this process repeats, we have that all the weights and the biases are adjusted at each epoch in order to increase the quality of the prediction. The metric that is used to evaluate

how the prediction is precise, is usually called the *loss function*. Generally, it is used the loss function that minimizes the *mean squared error (M.S.E.)*. So, for each epoch we have that after that the adjustment of the weights and biases is completed, the mean squared error is calculated, and then the weights and biases are adjusted again accordingly, to make sure that in the next epoch the M.S.E. is reduced. If the epochs are few, we aren't able to properly train the model, as the loss could be too high. In general, a high number of epochs is preferred, but if we exaggerate, we can get an extreme overfitting behaviour, which doesn't allow the neural network to get its overall trend.

Another important parameter to choose carefully is the *learning rate*. It tells the neural network how much to modify the biases and weights in response to the previously calculated loss. It is important to choose this parameter correctly because the loss of the model can even increase, which is of course not desirable. In fact, as the number of epochs increases, during the training, we want the loss to decrease. If, for example, the learning rate is too small, to achieve a significant decrease of the loss we have to increase the number of epochs significantly. On the other hand, if the learning rate is too big, the loss could result in an unstable behaviour or even worse it could increase.

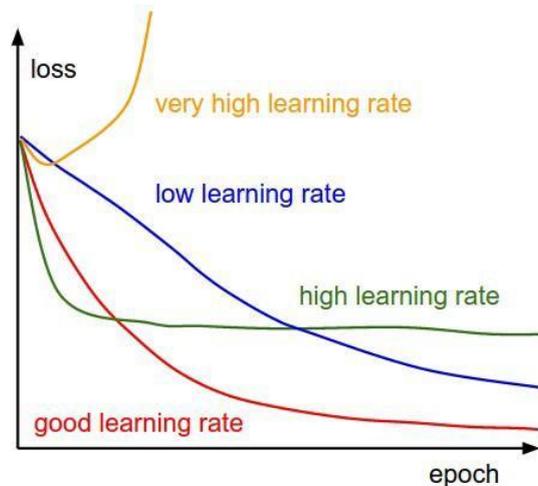


Figure 16, Loss behaviour as a function of the number of epochs depending on the learning rate

One last parameter to be kept in mind is the *batch size*, that is the number of rows of the database that are passed through to the whole structure of the network at one time. It can range from 1 to the full database. Generally, it is observed that, if the batch size is as big as the whole database, the prediction is not precise enough, but at the same time having it too small can still cause indecent predictions. To proceed correctly we must first set the batch size to a large number, even the whole database. Then, a gradual reduction has to be performed until we see that the loss doesn't decrease anymore. Then the batch size that produces the smallest loss, is the one to be chosen. Even though the tuning of this parameter is tedious, this one is one of the least important. It must be reminded

that we can play with the *number of epochs*, the *learning rate* and the *batch size* as much as we want, but the quality of the data inside the database is the one that majorly determines whether a neural network model is good or not.

#### 4.4 The activation functions

Before it was anticipated that the activation function can be chosen arbitrarily for each layer. This activation function modifies the manner in which the neural network processes the data. There are several types of activation function: **Linear**, **Sigmoid**, **Rectified Linear Unit (ReLU)** and **Hyperbolic Tangent (tanh)**:

##### 1) Linear Function

The output of the activation function  $f$ , is just given by the sum  $g_i$  itself:

$$f(g_i) = g_i$$

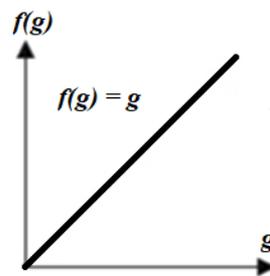


Figure 17, Linear activation function

##### 2) Logistic Function (Sigmoid)

The logistic function outputs the following shape, which reminds of the hyperbolic tangent function even though, unlike the tanh function, the image of this function is bounded between 0 and 1:

$$f(g_i) = \frac{1}{1 + e^{-g_i}}$$

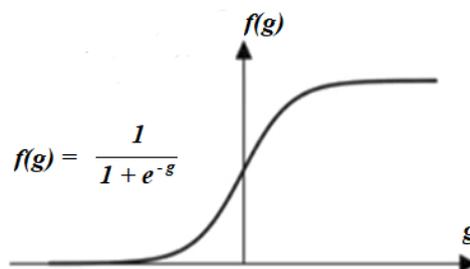


Figure 18, Sigmoid activation function

### 3) Rectified Linear Unit Function (ReLU)

This kind of activation function is one of the most popular. It is equal to the *sum* function  $g_i$  when  $g_i$  is greater than zero, and it's zero if  $g_i$  is lower or equal to zero.

$$f(g_i) = \max(0, g_i)$$

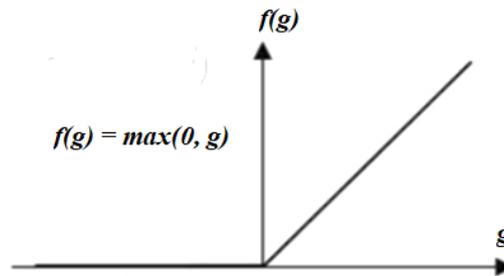


Figure 19, Rectified Linear Unit (ReLU) activation function

### 4) hyperbolic Tangent Function

Along with the ReLU, it's one of the most popular activation functions. It is like the sigmoid function but scaled by 2 and shifted down by 1, therefore lying between -1 and +1:

$$f(g_i) = \tanh(g_i)$$

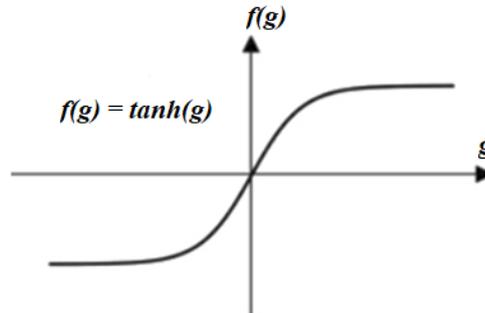


Figure 20, The hyperbolic tangent (Tanh) activation function

Now that a more precise insight on how neural networks work has been given, it is a good practice to be more specific about the real workflow that is behind a neural network epoch.

The input layer takes the information from the training database transferring it to the first layer of the neural network (dense hidden layer). The input layer has as many nodes as the number of features used to predict the labels. Then it is assigned a random weight  $w_i$  to each input feature  $x_i$  (this represents the part of the equation  $w_i \cdot x_i$ ) thanks to the connections that are present between each input node and each neuron of the first layer. Next the bias  $b_i$  is added ( $g_i = w_i x_i + b_i$ ) building the sum function  $g_i$ . The sum function  $g_i$  is fed to the chosen activation

function  $f_a$ , for the first hidden layer.  $f_a$  tells the network which input node is used to extract the feature. This is repeated as many times as the number of hidden layers, in which each one of them has its own activation function, up until the output layer is reached. The output layer must have as many nodes as the labels that we want to predict. This latter one has also its own activation function. Generally, the output layer has a linear activation function, because the other ones would act as a filter by distorting the final information. Then the output is delivered and the weights are adjusted to minimize the error for the next training epoch. This happens because the model compares the predicted output labels, with the original labels of the database; in this way we are able to determine the accuracy. This process is repeated as many times as the number of epochs, making sure to minimize the loss based off the chosen loss function. In general, the loss function is chosen between the predetermined ones that are available inside the Python library “*Tensor Flow*”.

However, as it will be shown later in this work, we can also create our own customized loss function. This will give rise to the ***Physics Informed Neural Networks (P.I.N.N.)***, that in our case is one of the best solutions that we can adopt to “inject”, inside the network, the physical and phenomenological behaviour that is involved in fatigue problems. However, as a beginning, we can consider one of the simplest cases of neural network: the ***Feed-Forward Neural Network (F.F.N.N.)***.

## 5. The Feed-Forward Neural Network (F.F.N.N.)

When it comes to the types of neural networks, there are several kinds of them. However, in this work, only two types will be considered: the *Feed-Forward Neural Network* (F.F.N.N.) and the *Physics Informed Neural Network* (P.I.N.N.). In this chapter the first one will be analyzed. The F.F.N.N. is the simplest model of neural network, in which data can travel only in the forward direction, so from the input layer to the output layer. While doing so, there is no information traveling backwards, as it would happen in other types of networks.

### 5.1 Preliminary tuning

As a starting point, we need to determine a useful structure of the neural network and, at the same time, have an initial tuning of the machine learning parameters, that is optimal for the required work.

The code is built as follows:

1. The database is retrieved, but it is built in a simplified manner, with respect to the one shown in *Table 1*. In fact, we are only considering as features the *orientation, scan speed, power, hatch distance, layer thickness* and the *stress amplitude*. As labels we are considering the *number of cycles to failure*, but it has been performed the logarithm with base 10 ( $\log_{10}(N_f)$ ).
2. Then, in order to grant an optimal training, the database needs to be normalized, such that the values range from small negative values to small positive values (from -3 to +3 circa). To accomplish this, it was computed the so-called Z-score of each column on the whole database, that is simply given by the following formula:

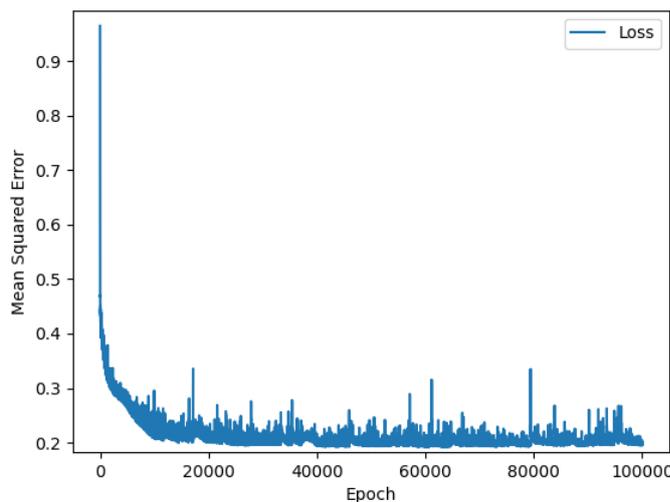
$$Database_{Normalized} = \frac{Database - Database_{Mean}}{Database_{standard\ deviation}}$$

The mean and the standard deviation of the database are computed column-wise, meaning that we have a mean and a standard deviation for each feature and for the label.

3. The structure of the neural network is defined. In this case, the input layer has 6 nodes. Then a line of code has to be written for each hidden layer, in which it is needed to specify the activation function and the number of neurons. It is important to experiment with the structure of the neural network in order to get an optimal decrease of the loss. The output layer has, of course, just one node because we need to predict one label.
4. The three machine learning parameters *epochs, learning rate* and *batch size*, are defined. In order to find an optimal combination, it is needed to experiment several times.

5. The model is trained from the data coming by the normalized database. Here the loss function is specified which, for the feedforward neural network, is the mean squared error metric (M.S.E.).
6. The loss calculated at each epoch is plotted against the number of epochs. This is done to understand how the neural network is reacting. The diagram that is obtained is the one similar to *Figure 16* at *section 4.3*.

After extensive experimenting with the machine learning parameters and the neural network structure, an optimal combination of number of neurons was found (*Table 2*) and it was validated by the following graph (*Figure 21*).



*Figure 21, M.S.E. as a function of the number of epochs*

Input layer	6 Nodes
Hidden Layer 1	100, ReLU
Hidden Layer 2	75, ReLU
Hidden Layer 3	50, ReLU
Output Layer	1, Linear

Batch Size	500
Learning Rate	0.001
Epochs min	500
Epoch max	10 000

*Table 2, preliminary Hyper-parameters*

The training could have been stopped earlier, for example after 1000 epochs. However, to make sure that the chosen combination was the right one, the training was carried out up to 100 000 epochs. Indeed, it can be seen that the learning rate was chosen correctly, because we have a steep decrease at the first 500 epochs and an overall steady phase after those ones. The plot resembles the one labeled as “good learning rate” in *Figure 16* and this is a sign that what we have obtained is a good starting point. For starters, the number of epochs is not to be chosen as a fixed value but as a range. A precise value will later be chosen in the validation, by plotting the predicted fatigue curve on top of experimental points, that are sharing the same process parameters.

The loss vs. epoch graph is just plotted once as a beginning verification of the chosen parameter. In the next sections it will not be considered anymore.

## **5.2 Preliminary validation**

Finally, once the neural network prediction has been produced after the training of the model, we can extract a part of the prediction, to see if the model is correctly predicting the points on the fatigue diagram. To this end, a random sub data set

has been selected with a dozen points. On top of this one a fatigue curve, produced by the neural network, is plotted. To accomplish this, the data set “Gong2015 SLM-MP 4” has been randomly selected (Figure 22 and Table 3).

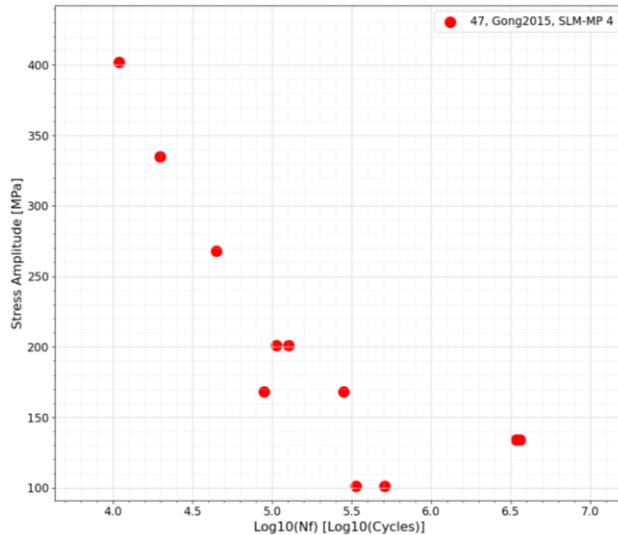


Figure 22, Gong2015 SLM -MP 4 dataset

	$\sigma_a$ [MPa]	$N_f$ [ $\log_{10}(\text{Cycles})$ ]
1	402	4.04
2	335	4.29
3	268	4.65
4	168	4.95
5	201	5.02
6	201	5.10
7	168	5.45
8	101	5.52
9	101	5.71
10	134	6.54
11	134	6.56

Table 3, Gong2015 SLM-MP 4

Then, another database, that is needed to evaluate the neural network predicted  $N_f$ , is built having as columns, the same process parameters of this dataset and the stress amplitude that is ranging from the maximum to the minimum stress amplitudes of the set (402 MPa to 101 MPa) with steps of 1 MPa (Table 4).

	Orientation [°]	Scan Speed [mm/s]	Laser Power [W]	Hatch [mm]	Layer Thickness [μm]	Stress Amplitude [MPa]
1	90	120	1260	0.1	30	402
2	90	120	1260	0.1	30	401
...	90	120	1260	0.1	30	...
302	90	120	1260	0.1	30	101

Table 4, Neural network preliminary evaluation dataset

Of course, the number of cycles is not present because it’s what we are interested in evaluating. After having built this evaluation database, since the neural network is stored in a normalized state, to grant continuity between the two databases, we need to normalize also this latter one. This time we still have to use the mean and the standard deviation of the full database and not the one of the smaller evaluation database.

$$EvaluationDB_{Normalized} = \frac{EvaluationDB - Database_{Mean}}{Database_{standard\ deviation}}$$

Finally we can predict the number of cycles to failure of the neural network via the function *tensorflow.predict*, that outputs  $N_f$  in a normalized form ( $N_{f,NN,norm}$ ). To bring it back to its standard form, we have to apply the inverse formula, but this time using only the mean and the standard deviation that have been calculated with respect to the column of the number of cycles  $N_f$ .

$$N_{f,NN} = N_{f,NN,norm} \cdot N_{f,std.dev.} + N_{f,mean}$$

Finally, is it possible to plot (*Figure 23*) the vector of predicted number of cycles vs. the vector of stress amplitude of the database illustrated in *Table 3*.

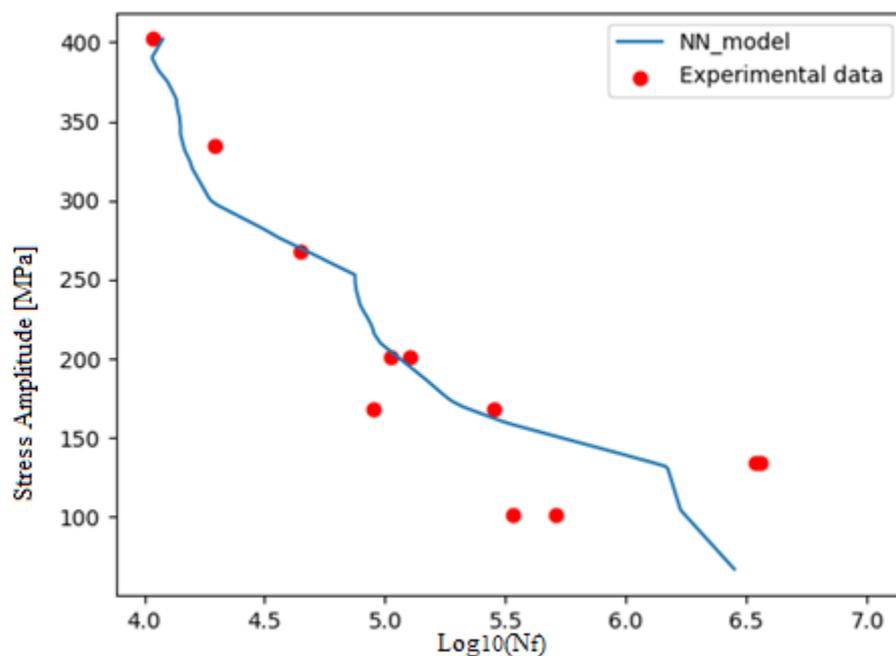


Figure 23, Gong2015 SLM-MP 4 Preliminary result

As a first result, it can be stated that we are going in the right direction even though there are a lot of things to be improved. The following picture has been obtained with a low number of epochs. It is important to investigate what happens to the fatigue curve if we modify the number of epochs.

### 5.3 Modifying the number of epochs

As it was anticipated before, if we use a low number of epochs, a median behavior is obtained. While, if we exaggerate, we get more accurate predictions, but the neural network overfits the data present in the database, which is not an optimal behavior, especially if we consider the typical shape of the fatigue curves. For instance, to understand which number of epochs is optimal, the training has been run several times with an increasing number of epochs each time. The test was

carried at 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 5000, 10 000 and 20 000 epochs. Special attention was brought to the range near 500, as it was deduced that optimal results can be obtained in those regions. *Figure 24* illustrates this concept, passing from 200, 900, 5000 and 20 000 epochs:

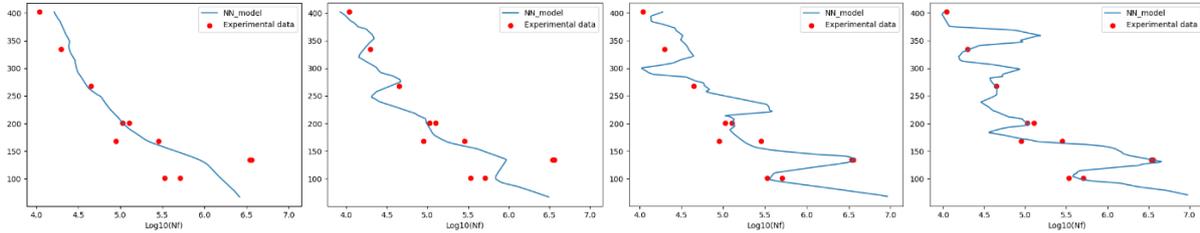


Figure 24, Increasing the number of epochs results in an overfitting behaviour

The behavior at 200 epochs is much more preferable as it resembles more the one typical of fatigue curves, even though there is still a lot to improve. At 20 000 epochs, the curve hits all the points precisely, but an overfitting behavior must be avoided. On the shape of fatigue curves, generally it is favorable to have curves that are strictly decreasing with a concavity facing upwards. Even better if they are bi-linear. This behavior is needed to represent the phenomenological behavior of fatigue problems. *Figure 25* shows a typical fatigue curve.

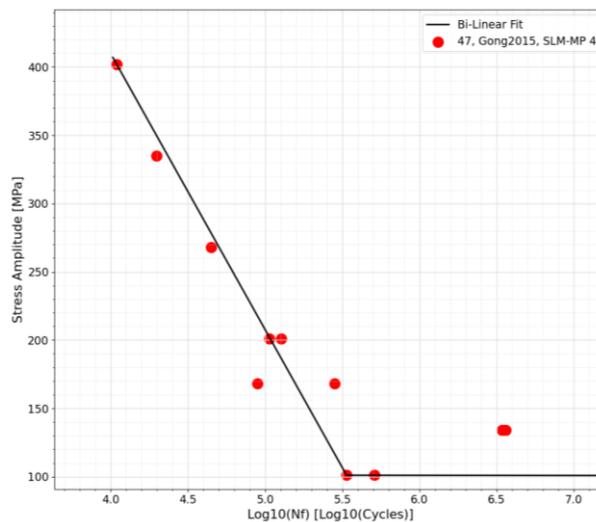


Figure 25, Typical Bi-Linear behaviour of fatigue curves

### 5.4 Removing the validation set from the training database

Usually in machine learning problems, it’s a good practice to remove the dataset that we want to validate from the training database. This is generally done to ensure that the method is solid and stable and to understand the right direction to take. Unluckily, after having performed this operation, the quality of the predictions drastically decreased as shown in *Figure 26*.

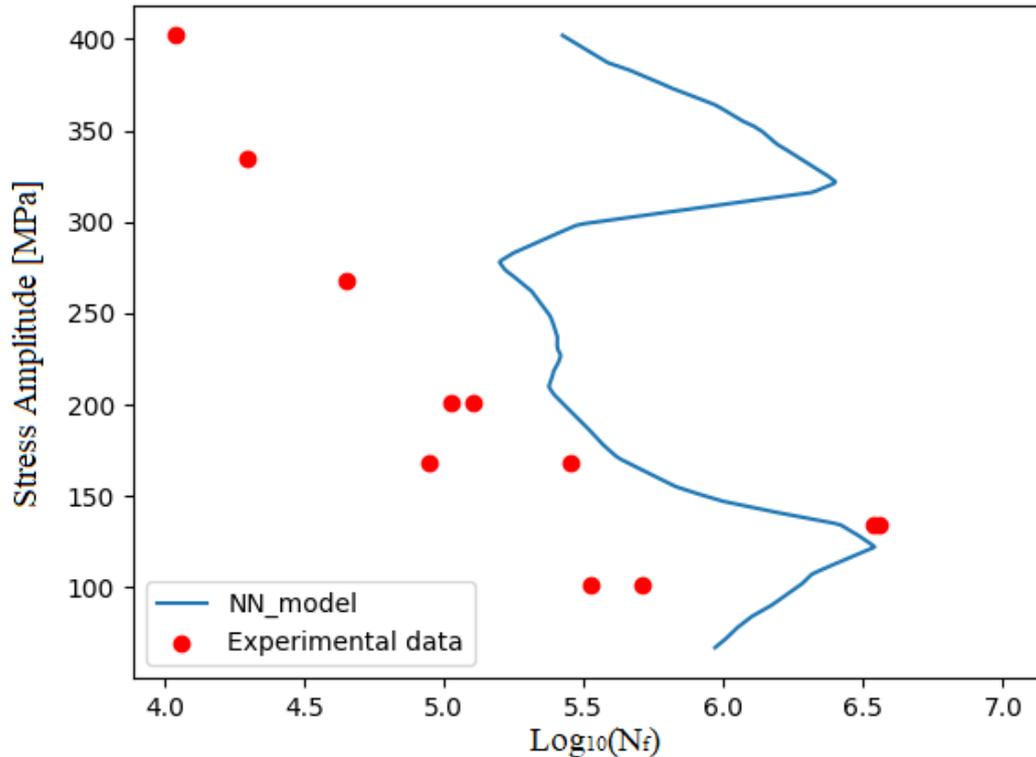


Figure 26, Preliminary fatigue curve after removing the dataset from the training database

The aspects to be improved are still significant:

- The curve is not strictly decreasing.
- The prediction is nearly completely wrong.
- The curve is not smooth, but it's made of broken lines.
- The behaviour is not the one typical of fatigue problems.

At least the prediction falls in the range of the experimental data, meaning that we are starting from a decent point. To solve the previously listed issues, we can start enriching the database with the information on the thermal and surface treatments, as well as tweaking the process parameters, since it is observed that the quality of the predictions is mostly determined by the content of the database and on how the database is built, rather than the neural network structure or the machine learning parameters themselves. In fact, the structure and the set of machine learning parameters shown in *Table 3*, were confirmed to be working, even though the database was enriched gradually.

## 6. The Physics Informed Neural Network (P.I.N.N.)

To make sure to have the best possible prediction, it is a good practice to remove the validation set from the training database, to see if the points to be predicted are still fit by a nicely shaped curve. In the last section the prediction was not good, because there are several things to be solved. In this section, three validation sets will be used instead of just one. This has been done to make sure that what has been obtained in the region of the data set “*Gong2015 SLM-MP 4*” is not an exception but something that is a characteristic of the whole database. For all the three cases, the removal of the validation set from the database before training has been performed to grant continuity of the method.

### 6.1 The correlation matrix

But before proceeding with any kind of important modification to the code, it is usually a good practice to study the correlation between the features towards that label  $N_f$ . In fact, with this strategy, we can see which feature has a high predictive power and which one has a low one, such that we can perform modifications that will improve the quality of the predictions. To this end the *correlation matrix* of the whole database is performed in Python via the command `.corr()`. As it can be seen in *Figure 27*, the matrix gives a coefficient that relates each column of the database to all the other columns (even to the same column, which corresponds to a coefficient of 100 %).

	Orientation	Power	...	Nf
Orientation	100.000000	12.518486	...	14.630142
Power	12.518486	100.000000	...	-9.508145
Hatch	-17.073513	21.369463	...	-18.934492
Speed	27.976791	6.860163	...	6.765047
Layer_thickness	-6.138733	49.647117	...	8.098900
Nf	14.630142	-9.508145	...	100.000000
Fatigue_limit	3.201712	51.744686	...	-38.367279

Figure 27, Correlation matrix of the training database

The column that we are interested in analyzing is the one relative to the label  $N_f$ . Indeed, we can see that the feature *laser power*, *laser scan speed* and *layer thickness* have a low predictive power with respect to the other ones. For sure something can be done to improve the quality of the predictions. In fact, as it was explained in *Section 2*, it has been observed that laser power and laser scan speed have a strong correlation between each other. Therefore, if the speed is too high

and the power is too small, we don't have enough time to melt the powder. Vice versa if we have too much power and time to melt the powder, the multiple layers that were previously formed can be remelted again and again, ruining the quality of the finished part. These observations, along with the results produced by the correlation matrix, are a great suggestion to produce a *feature cross*. A feature cross is simply a combination of two or more columns of the database, that is performed via multiplications or divisions. In our case there are not many combinations to be tried, because we have just two columns to combine. So, we can produce the correlation matrix of all the possible combinations that can be made between these two columns and see which one of them gives the highest predictive coefficient with respect to the label  $N_f$ . We just have 3 possibilities:

1. *Laser Power / Laser Scan Speed*
2. *Laser Power · Laser Scan Speed*
3. *Laser Scan Speed / Laser Power*

### 6.1.1 Laser Power / Laser Scan Speed

As *Figure 28* shows, we can see that, if we are performing the element wise division of the laser power feature by the laser scan speed feature, we are getting a correlation coefficient with respect to  $N_f$  that is not that much higher with respect to the features considered alone. In fact, before we had that for the power the coefficient was **9.51%** and for the scan speed it was **6.77%**. While now, not only do we have one column less to predict the label, but the coefficient of the feature cross is almost identical to the highest of the two. For sure **9.62%** is not a significant improvement and it's not worth removing a column for such a small change. To make the feature cross worth, its correlation coefficient has to increase significantly, otherwise we'd better keep everything as it was before.

	Orientation	Power/Speed	...	Nf
Orientation	100.000000	-9.857585	...	14.630142
Power/Speed	-9.857585	100.000000	...	-9.623989
Hatch	-17.073513	21.358660	...	-18.934492
Layer_thickness	-6.138733	42.363771	...	8.098900
Nf	14.630142	-9.623989	...	100.000000
Fatigue_limit	3.201712	15.759264	...	-38.367279

Figure 28, Correlation matrix with Power/Speed feature cross

### 6.1.2 Laser Power · Laser Scan Speed

Going forward with the second attempt, we can see that this one produces a coefficient that is drastically worse than the first one. For sure, such a low number (**1.19%**) must be avoided, because it doesn't make the effort worth (*Figure 29*).

	Orientation	Power*Speed/1000	...	Nf
Orientation	100.000000	23.903390	...	14.630142
Power*Speed	23.903390	100.000000	...	-1.194269
Hatch	-17.073513	-2.274033	...	-18.934492
Layer_thickness	-6.138733	16.757745	...	8.098900
Nf	14.630142	-1.194269	...	100.000000
Fatigue_limit	3.201712	38.512735	...	-38.367279

Figure 29, Correlation matrix with Power \* Speed feature cross

### 6.1.3 Laser Scan Speed / Laser Power

Last but not least, we still have the reciprocal of the first attempt to be tried. Printing the correlation matrix, we can finally see that the effort has been paid, because the correlation coefficient is significantly higher (**13.54%**) than the one of that feature columns separated.

	Orientation	Speed/Power	...	Nf
Orientation	100.000000	18.849969	...	14.630142
Speed/Power	18.849969	100.000000	...	13.535605
Hatch	-17.073513	-40.580158	...	-18.934492
Layer_thickness	-6.138733	-37.105108	...	8.098900
Nf	14.630142	13.535605	...	100.000000
Fatigue_limit	3.201712	-33.461753	...	-38.367279

Figure 30, Correlation matrix with Speed/Power feature cross

From now on the training database will be considered with this modification. *Table 5* resumes the study that has been just shown:

Feature	Correlation coefficient
<i>Power</i> and <i>Speed</i> alone	<b>9.51%</b> and <b>6.77%</b>
<i>Power / Speed</i> feature cross	<b>9.62%</b>
<i>Power · Speed</i> feature cross	<b>1.19%</b>
<i>Speed / Power</i> feature cross	<b>13.54%</b>

Table 5, Feature cross attempts and their correlation coefficients

## 6.2 Enriching the database of the P.I.N.N.

As it was stated at the beginning, usually in machine learning problems, it's good practice to tune the network with a simplified version of the full database. But once a preliminary tuning is performed, it's time to improve the predictive power by including the full database. This procedure has been carried through gradually, without including all the database at once. This was done to see if the inclusion of more features could have degraded the quality of the prediction. At the end it didn't. The shape of the database is identical to the one of *Section 3*, the only difference being the addition of the feature cross *Speed / Power* (*Table 6*).

	A	B	C	D	E	F	G	I	J	K	L	M	N	O	P	Q	R	S	T
1																			
2																			
...																			
i																			
...																			
N																			

A	Article Subset Name	L	Machined Boolean
B	Article Name	M	Sand Blasted Boolean
C	Subset Global Code	N	E.D.M. Boolean
D	Build Orientation [°]	O	L.S.P. Boolean
E	Scan Speed / Power [mm/s·W]	P	S.P. Boolean
F	Hatch Distance [mm]	Q	S.M.A.T. Boolean
G	Layer Thickness [μm]	R	Surface Polishing Boolean
I	Annealed Boolean	S	Stress Amplitude (at R= -1) [MPa]
J	Annealing Temperature [°C]	T	Fatigue Life $N_f$ [Cycles]
K	H.I.P. Boolean		

Table 6, Updated training database with feature cross

An important concept to keep in mind is the fact that, since the training database needs to be normalized, as it was done in *Section 5*, it's important to perform these additional operations to avoid problems during the training:

1. Store in a separate database all the boolean columns (*Annealing*, *H.I.P.*, *Machined*, *Sand Blasted*, *E.D.M.*, *Laser Shot Peened*, *Shot Peened*, *S.M.A.T.* and *Polished*)
2. Remove them from the training database
3. Normalize the Training database, with the same procedure of *Section 5.1*
4. Re-insert all the boolean values

If this is not done, since the boolean values contain zeros and ones, the chances of getting a *Not-a-Number* (NaN), are very high and the training database is not usable in its normalized state. The same holds for the database that is used to evaluate the neural network prediction (same procedure as in *Section 5.2*, but with this caveat to be done).

If we straight toss a rough prediction with the enriched database and the feature cross that has been performed before, we can see from *Figure 31*, that the quality of the prediction of the data set “*Gong2015 SLM-MP 4*” (with removal of this set from the training set), has significantly improved with respect to before (refer to *Figure 26* on *Section 5.4*).

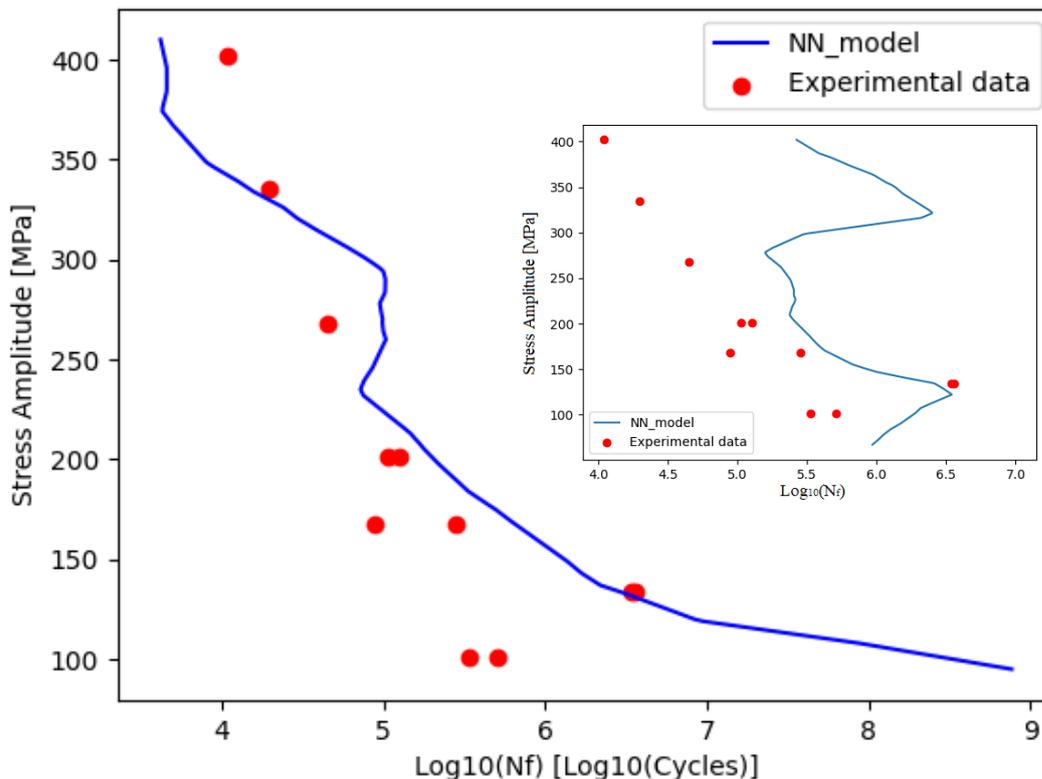


Figure 31, Fatigue curve after enriching the database

However, even though the quality is much better, we still have several problems to solve. In fact, the curve is not strictly decreasing and has still a broken line shape. Indeed, it is not still resembling the classical shape of fatigue curves. Recalling the list of things that needs to be improved from *Section 5.4*:

1. The curve is not strictly decreasing. (*Not solved*)
2. The prediction is nearly completely wrong. (*Solved*)
3. The curve is not smooth, but it's made of broken lines. (*Not solved*)
4. The behaviour is not the one typical of fatigue problems. (*Not Solved*)

### 6.3 Implementing the custom loss function

To solve the problem of the broken line fatigue curves, we can simply straight implement, as activation function of all the layer of the neural network, the *hyperbolic tangent*. This latter one is generally responsible for smooth behaviors. All the neural network parameters, the number of nodes and layers are kept the same. By simply changing the text from “ReLU” to “tanh”, we can easily achieve this change. Indeed, from *Figure 32*, we can see that we do not have broken lines anymore, as the curves are smooth.

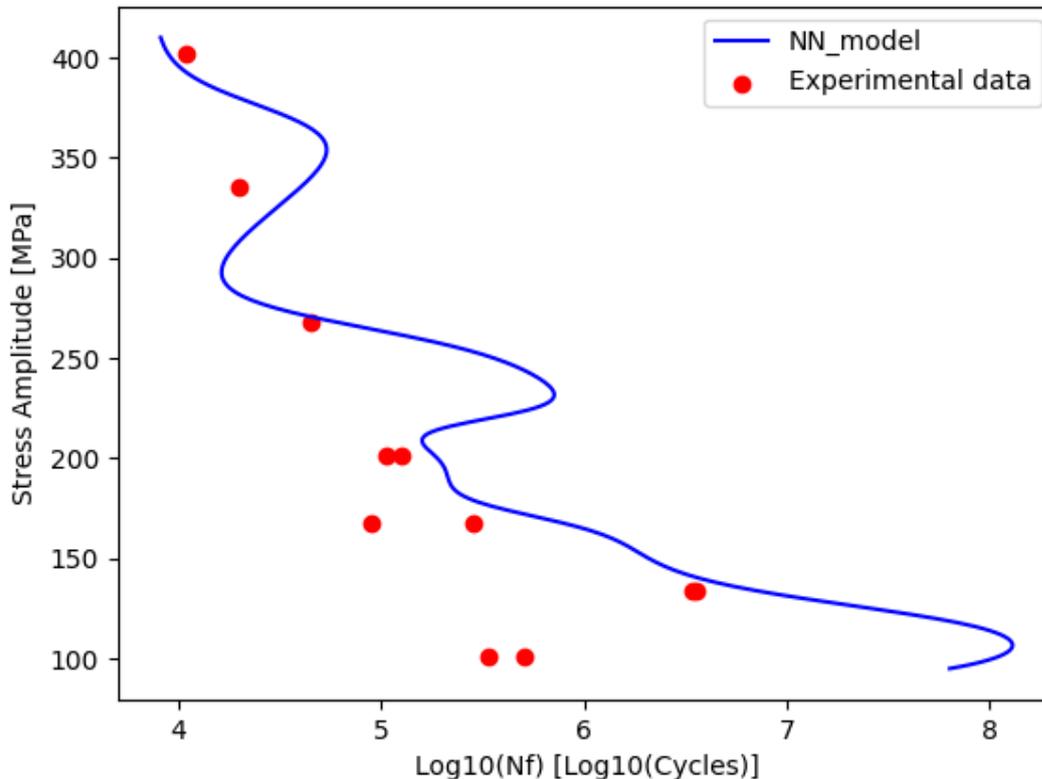


Figure 32, Fatigue curve after using the Tanh activation function

With this change, point 3 of the previous list has been solved. We still need to manage to achieve a strictly decreasing behavior, as we cannot afford to have this back-and-forth behavior. Fatigue curves must be strictly decreasing, since, through the years, it was observed that if we want a component to survive more (high number of cycles) we must decrease the stress amplitude to which the piece is subjected; vice versa if we want the mechanical part to withstand a higher stress amplitude, the piece will last less (low number of cycles). To this end we can inject the physics of the problem by considering the 1<sup>st</sup> derivative. In fact, if we are able to impose that the first derivative must be always negative, the previous problem is solved. This quest introduces us to the *Physics Informed Neural Network* (P.I.N.N.). As the name suggests we are informing the neural network, by telling it the physics that lies behind the problem that we want to predict. As it was stated before, if we can put a strictly decreasing behavior inside the

training, we are automatically taking into account the physics of fatigue problems. To achieve this, a custom loss function must be used instead of the loss metric “mean squared error loss function”, because this one considers only the precision of the model, but not the way the model achieves this precision. In poor words, this loss function is not able to dictate the shape of the fatigue curve. While compiling the neural network via the command “.compile”, the loss function to be specified is named “*customLoss*”. This allows us to create our own customized loss function that allows to develop the behavior that we are interested in.

### 6.3.1 How the curves are computed

To compute the curves a simple, yet effective strategy is used. A set of so-called sampling points is created. Their role is to sample the multidimensional space in which the dataset, that has to be evaluated, is located before the training. In fact, the number of cycles to failure  $N_f$ , it's a function of all the 15 features (process parameters, thermal treatments, surface treatments and stress amplitude). The aspect that must be taken into account, is that all the process parameters and the treatments of the set that has to be evaluated, are constant in this multidimensional space, while the only thing that varies is the stress amplitude.

For example, evaluating the set “Gong2015 SLM-MP 4” (again with its removal from the training database to ensure the strength of the method), that has the following features (Table 7), the minimum and the maximum stress amplitude are searched and a vector of 1000 points is created between these two boundaries.

	D	E	F	G	I	J	K	L	M	N	O	P	Q	R	S
1	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	402
2	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	401.7
...	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	...
i	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	315.3
...	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	...
1000	90	10.5	0.1	30	0	20	0	1	1	0	0	0	0	1	101

D	Build Orientation [°]	L	Machined Boolean
E	Scan Speed / Power [mm/s·W]	M	Sand Blasted Boolean
F	Hatch Distance [mm]	N	E.D.M. Boolean
G	Layer Thickness [µm]	O	L.S.P. Boolean
I	Annealed Boolean	P	S.P. Boolean
J	Annealing Temperature [°C]	Q	S.M.A.T. Boolean
K	H.I.P. Boolean	R	Surface Polishing Boolean
		S	Stress Amplitude (at R= -1) [MPa]

Table 7, Updated Neural network evaluation database

Of course, this procedure has to be done with the normalized database, so the example values reported in the table above are normalized with the usual procedure. Once this temporary database is created, it has to be converted into a tensor flow variable since it is going to be fed inside the custom loss function.

Basically, the idea that lies behind this procedure is to create a plane in a 16-dimensional space, with 14 dimensions that are constant (all features except the stress amplitude that is the only one varying from the maximum stress to the minimum of the considered data set). This plane is the usual S-N fatigue diagram. Then, once this virtual plane is created, we compute the derivative of the number of cycles with respect to the stress amplitude ( $dN_f/d\sigma_a$ ) and we penalize positive derivatives such that the negative derivatives, that corresponds to a decreasing behavior, are favored.

To accomplish this, the gradient is computed in the “TensorFlow” virtual space, and a **custom penalization function**  $f$  is inserted as a reference for the custom loss function. This function is responsible for making the code understand that we want to reduce positive derivatives while favoring negative derivatives.

$$f = \frac{-\frac{dN_f}{d\sigma_a} + \left| \frac{dN_f}{d\sigma_a} \right|}{2}; \quad g = \frac{dN_f}{d\sigma_a}$$

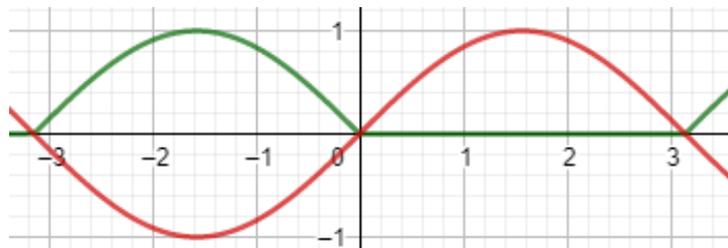


Figure 33, The custom penalization function

If, for example, we have that the derivative (**red**) has a determined behavior (in this case, for the sake of clarity, it has been chosen a sinusoidal behavior, just to explain, but it could have any kind of irregular shape), by performing the **green custom penalization function**  $f$ , we are zeroing the positive derivatives, while we are giving a positive weight to the negative derivatives. This means that the **custom loss function**, that will be explained in a while, will not consider positive derivatives while computing the MSE at each epoch. In few words only the negative derivatives will contribute to the precision of the neural network, because the positive derivatives are ignored and not taken into account. A little note to avoid confusion; the **custom loss function** is different with respect to the **custom penalization function**  $f$ . In fact, the **penalization function**  $f$  is used by the **custom loss function** to train the neural network, such that it respects the physic of the problem.

### 6.3.2 How the custom loss function works

The custom loss function can be recalled by the standardized name “*customLoss*” and it takes as input the standardized names  $y\_pred$  and  $y\_true$ . Those two are used to compute the mean squared error MSE, so they are just responsible of the precision of the neural network. In fact,  $y\_true$  represents the correct value that is inside the training database, while  $y\_pred$  is representing the prediction that is created at each epoch from the neural network.

The MSE is calculated through its usual standard formula:

$$MSE = \frac{\sum_{i=1}^N (y_{true,i} - y_{pred,i})^2}{N}$$

At the end, this *MSE* term, that is responsible of the overall precision of the neural network, is combined with the *penalization function f*. But before doing so, they are casted to a tensor flow variable via the command *tf.cast*; this allows them to be read as tensors inside the Python library *TensorFlow*, that is the one responsible of performing machine learning operations. The final loss that will be used by the neural network to evaluate the curves has another standardized name called *totalLoss* and it's simply given by the following formula [4]:

$$totalLoss = a \cdot f + b \cdot MSE$$

We can see that there are two coefficients *a* and *b* that are near *f* and *MSE*. Basically, their role it's to favor one of the two terms and to give them a weight. We have that the *a* term is going to favor more the precision that is relative to the derivative, making sure that it's a negative one. While the *b* term is the one responsible to give a weight to the overall precision of the network.

It is good practice to keep the sum of the two coefficients equal to 1, so  $a + b = 1$ . At this point a question may naturally arise: How do we choose these coefficients? Do we go by trial and error, inputting random values or is there any trend?

Initially it was tried to input *a* and *b* manually without applying any significant rule. Several unfulfilling curves have been obtained (*Figure 34, left*), leading to think that the whole procedure of the physics informed neural network was not a good strategy. But at the end there were sets of coefficients that produced excellent results (*Figure 35, right*). It has to be kept in mind that the number of epochs from now on will always be around 500, since it has been observed that this produces the best kinds of shape with the typical behavior of fatigue curves. A strategy to choose the values of *a* and *b* is surely needed to understand how these two coefficients influence the shape of the fatigue curve.

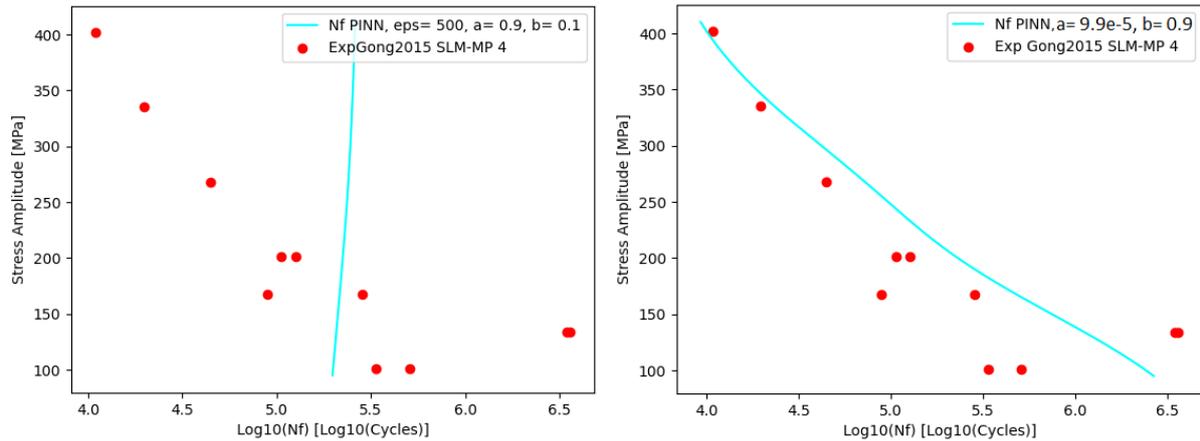


Figure 34, Change of behaviour with respect to the coefficients  $a$  and  $b$

### 6.3.3 Tuning the $r$ parameter

After playing a little bit with  $a$  and  $b$  with the dataset “*Gong2015 SLM-MP 4*”, it was decided to extend the evaluation to two other randomly chosen datasets: “*DuQian2020, set 1*” and “*Alegre2022, As Built*” (for each one of the three datasets, the training database had those points removed, to grant that the method is sufficiently solid). At the same time, it was chosen to adopt a strategy that had a quantifiable method. Since, changing each time manually these two coefficients was time consuming, it was decided to introduce a new parameter that is a combination of  $a$  and  $b$ : the  $r$  parameter. This latter one it’s simply the ratio of  $a$  and  $b$ . Afterwards,  $a$  and  $b$  were re-written as a function of  $r$  with trivial mathematical operations:

$$r = \frac{a}{b}; \quad a + b = 1; \quad a = \frac{r}{r + 1}; \quad b = \frac{1}{r + 1}$$

In this way, we can just change one value and the two coefficients will change automatically consequently. Again, to exclude any kind of randomness, it was decided to create a vector of values of  $r$ , that are ranging from the order of magnitude of  $10^{-12}$  to  $10^{12}$ . More precisely it is going from  $1 \cdot 10^{-12}$  to  $1 \cdot 10^{12}$ , alternating the coefficient in front of  $10^x$  with 1 and 5 (*Table 8*).

$r$	$1 \cdot 10^{-12}$	$5 \cdot 10^{-12}$	$1 \cdot 10^{-11}$	$5 \cdot 10^{-11}$	...	$1 \cdot 10^{11}$	$5 \cdot 10^{11}$	$1 \cdot 10^{12}$	$5 \cdot 10^{12}$
-----	--------------------	--------------------	--------------------	--------------------	-----	-------------------	-------------------	-------------------	-------------------

Table 8, All the possible values of the  $r$  parameter

This allows us to have a decently sized spectrum to investigate how the behavior of the fatigue curve changes with respect to  $r$ . In any case we do not have to worry about  $a$  and  $b$  because, as it was stated before, they are automatically determined by the previously imposed constraint ( $a + b = 1$ ).

The idea is to launch a training session for each of the three randomly chosen subsets for each value of the  $r$  parameter, to see which value of  $r$  is the one

producing the best results. However, in the first attempts, it was observed that with the same  $r$  parameter and the same settings, the obtained curves were a bit mismatched. In order to compensate for this behavior, the training for each dataset at each  $r$ , was repeated five times leading to get 5 curves and by performing the mean of these five curves, a better prediction was obtained. The hyperparameters and the structure of the neural network was the usual one and it was kept constant for each single launch (epochs = 500, learning rate = 0.001, batch size = 500, layer 1 = 100 neurons with tanh, layer 2 = 75 neurons with tanh and layer 3 = 50 neurons with tanh).

In addition, to have both a qualitative and quantitative description of the fatigue curves, the *root mean square error (R.M.S.E.)* was manually calculated between the fatigue curve (the one averaged on the 5 predicted curves) and the experimental points in the X direction, when the values of the stress amplitude were equal, thanks to a couple of nested *while loops*.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (N_{f,experimental,i} - N_{f,NN\ prediction,i})^2}{N}}$$

So, for each  $r$  and for each dataset, we have a root mean square error that is representative of the dataset and of the selected  $r$  parameter. This allows us to have a quantitative description of how good the prediction is. While, in order to have a qualitative idea of the curve, it is just needed to observe the plot of the average curve on top of the experimental points. Combining these two strategies will allow us to have the best possible choice of the  $r$  parameter, since it's not possible to have a customized value of the  $r$  parameter for each single data set, as it cannot be changed each time.

Just to have an idea on how many times the code has been automatically launched thanks to several nested while loops: we know that for each of the 3 data sets the training was launched 25 times (that's the length of the  $r$  parameter vector) and for each value of the  $r$  parameter the training occurred 5 times, to create a five curves average; that's a total of 125 trainings per dataset. To then compare the values of the root mean squared error, a mono-logarithmic plot was produced, having on the X axis the values of the  $r$  parameter, while on the Y axis the value of the *root mean square error (Figure 35)*.

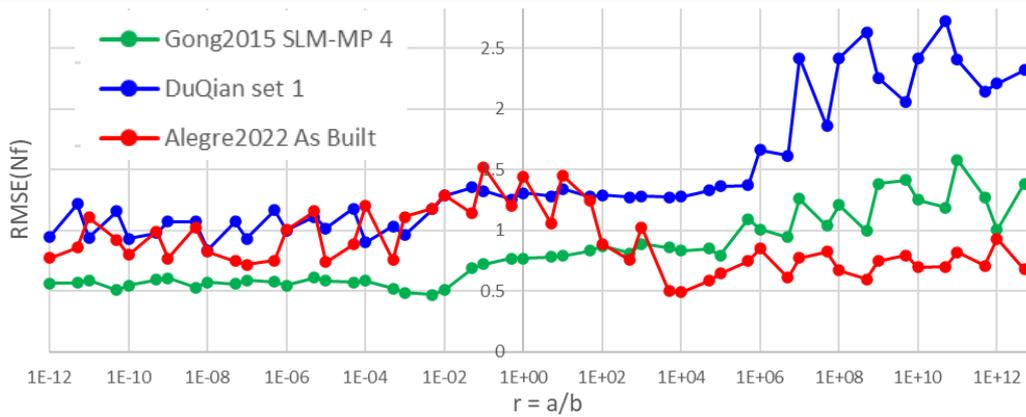


Figure 35, R.M.S.E.(Nf) as a function of r for the 3 analysed datasets

Having performed the mean on five curves was a good strategy to reduce the randomness that is present in each training. In fact, we can see that the curves in this plot are not so much fragmented. We can finally investigate them to look for a minimum value of the **RMSE** to have the best prediction.

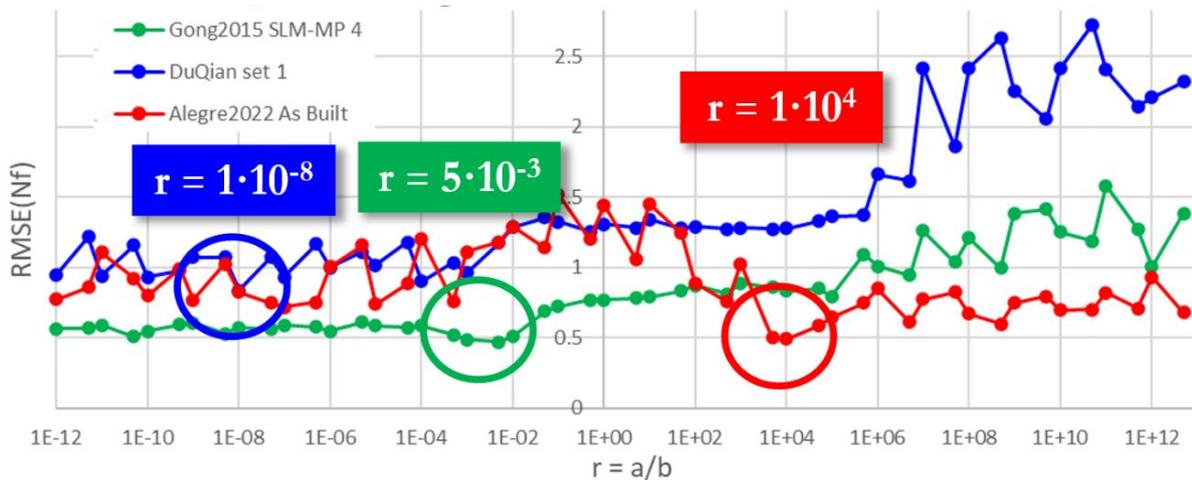


Figure 36, Sweet spots for the 3 datasets

If we look at the minimum value for each data set, we see that for each curve we have values that are not the same. Each dataset was expected to have the minimum in the same range, but disappointingly enough, this didn't occur. However, as it was stated before, this is just representing a quantitative numerical approach. We can still rely on the qualitative investigation, made possible by looking at the fatigue curves. The following figure (*Figure 37*) shows the fatigue curves that are relative to the  $r$  values highlighted in *Figure 36* (the black curve is the one averaged from the 5 purple ones).

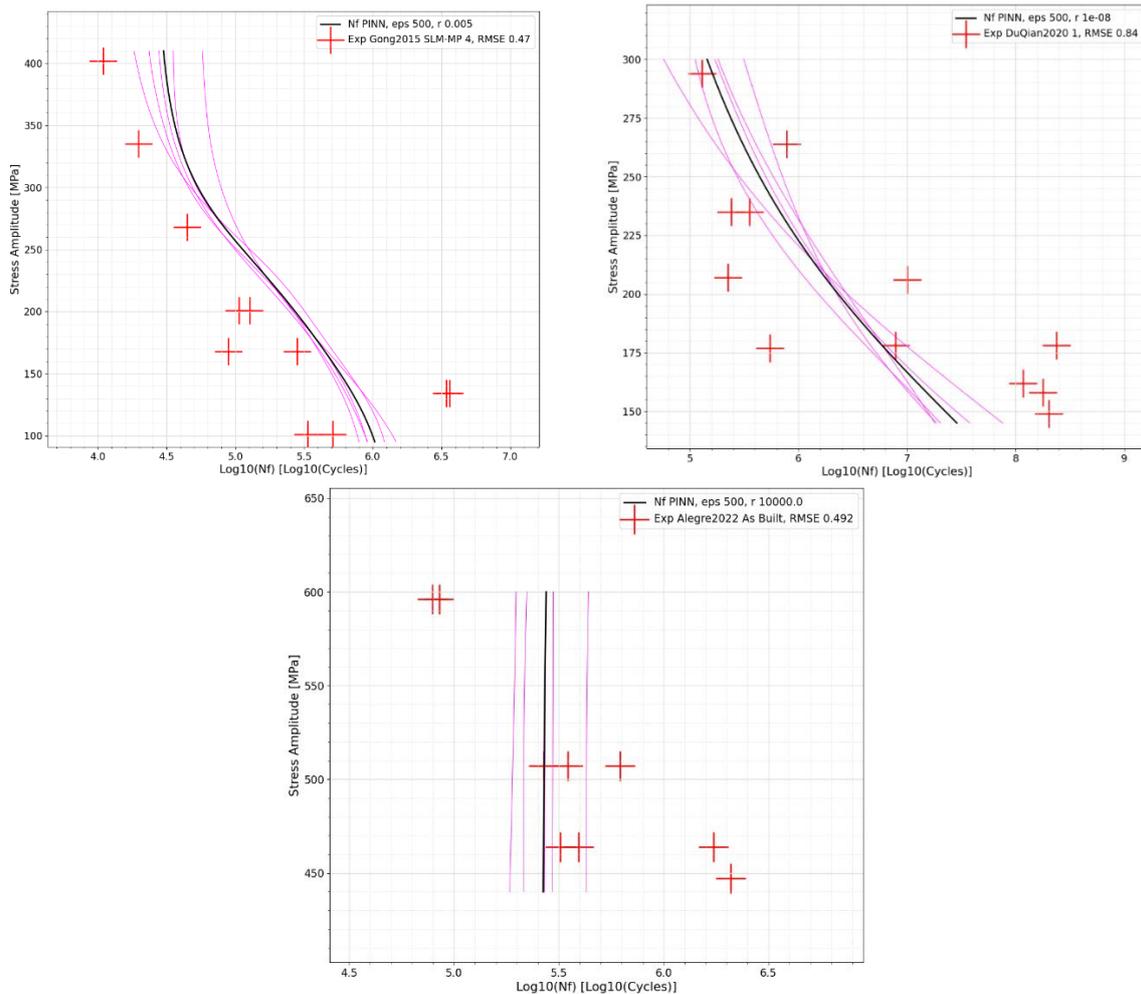


Figure 37, Fatigue curves corresponding to their sweet spots

Indeed, we can see that for the sets “*Gong2015 SLM-MP 4*” and “*DuQian2020, set I*”, as it can be verified by the legend present in the picture, the prediction is not bad and acceptable. While for the subset “*Alegre2022, As Built*” the fatigue curve cannot be called in such way: the lines are vertical leading to a completely useless prediction. The occurrence of this phenomenon is due to the fact that we didn't explicitly use, in an independent manner, the qualitative evaluation that is carried out by looking at the fatigue curves. In fact, for how helpful the numerical evaluation is, it can happen that the *root mean square error* calculated for each  $r$  is not representative of the goodness of the prediction. This happens because there are curves that are not strictly decreasing, or that are just vertical, that are giving us the best results in terms of *RMSE*, with respect to curves that are strictly decreasing. That's why it was useful to combine both the qualitative evaluation, by observing the fatigue curves, and the quantitative one, given by the *RMSE* value. Finally, if we look for a fixed value of  $r$  for all the three curves, since it was observed that for values of  $r$  that are between  $10^{-3}$  and  $10^{-8}$ , we get excellent predictions for all the 3 sets, we can choose  $r = 10^{-4}$  as a good candidate (Figure 38).

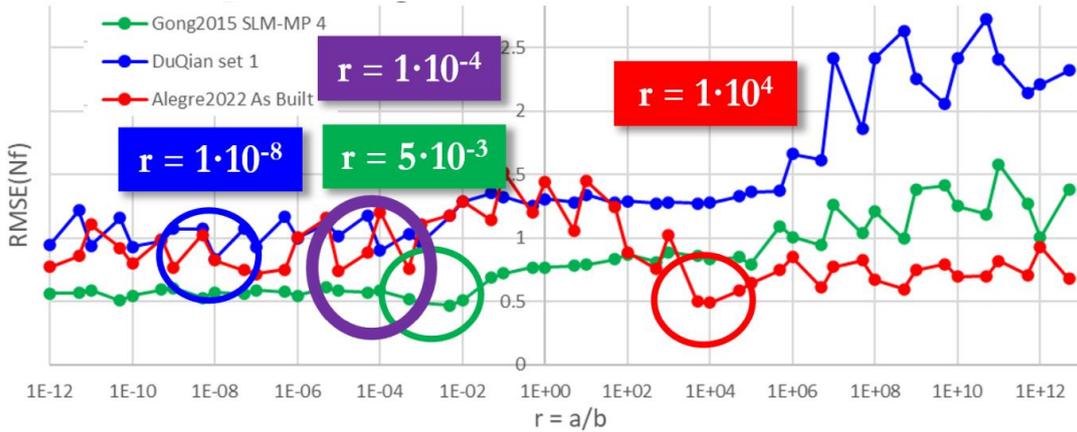


Figure 38, Final sweet spot choice

This choice was simply made thanks to both the qualitative and quantitative evaluation, by looking at the figures in the previously highlighted range (from  $10^3$  to  $10^{-8}$ ). Because with the **RMSE** we got a rough idea of the useful range, and by looking at the pictures we can validate this idea and make a final choice. *Figure 39* confirms what was told before:

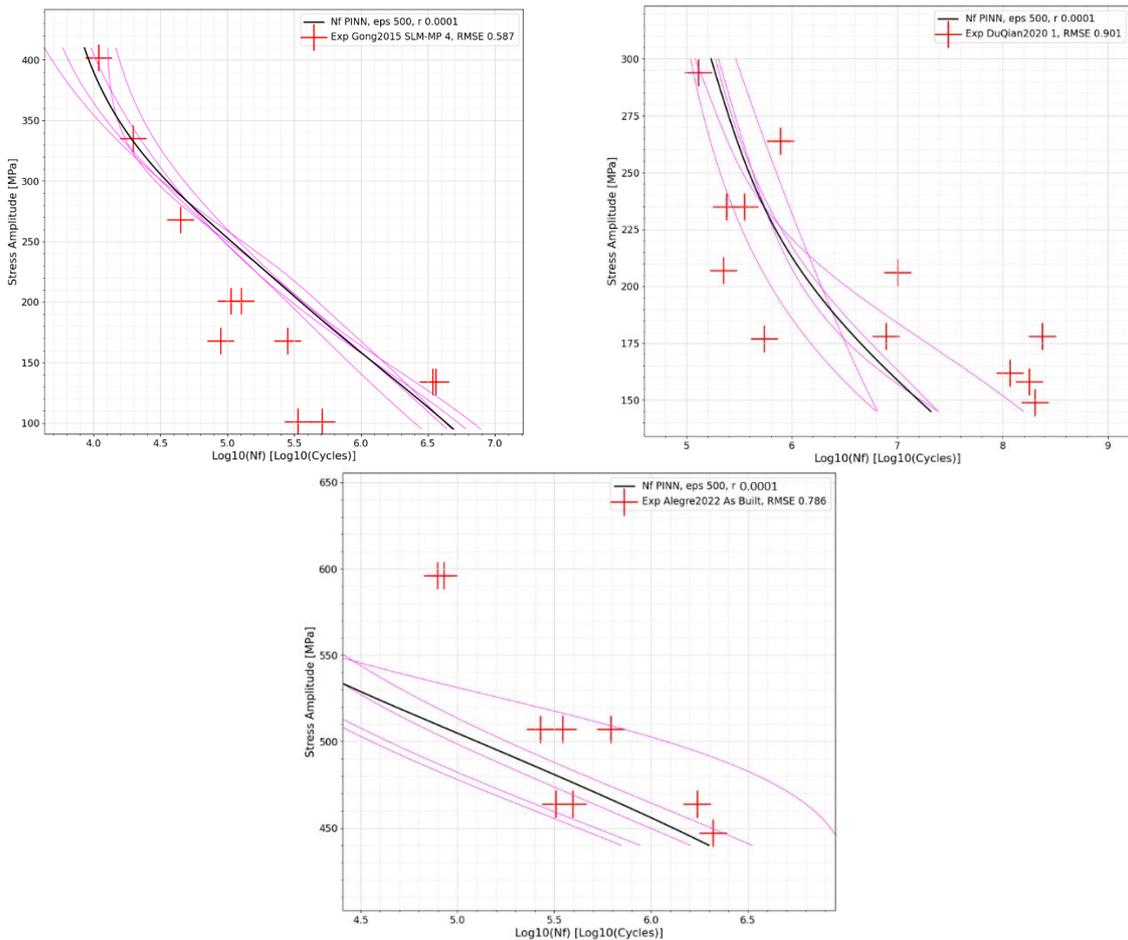


Figure 39, Fatigue curves corresponding the final choice of the  $r$  parameter

After this long study, it can be finally concluded that choosing  $r = 10^{-4}$ , leads to satisfactory results. In fact, for the sets “*Gong2015 SLM-MP 4*” and “*DuQian2020, set 1*”, as it can be verified in the pictures, the curve is nicely shaped. For the subset “*Alegre2022, As Built*”, we could have gotten better results, but even though the curve is miss centered, at least we got the final part of the diagram, and we are from the conservative side in the top part. It must be remembered though, that all the three subsets have been removed from the training, meaning that that curves are in the worst-case scenario. If the datasets were present inside the training database, their shape would have been much better. Indeed, we can proceed by evaluating what has just been told, as well as each single subset of the training database; this time using the training database in its entirety.

### 6.4 Global evaluation

Table 9 has been reported for clarity, to see how many subsets we have to validate. In total we have 76 of them.

Code	Article	Sub-Set Name	Code	Article	Sub-Set Name
0	DuQian2020 [5]	1	38	Gunther2018 [14]	Batch 1-3
1	DuQian2020 [5]	2	39	Gunther2018 [14]	Batch 2
2	DuQian2020 [5]	3	40	Gunther2018 [14]	Batch 3
3	DuQian2020 [5]	4	41	Eric2016 [15]	As Built
4	DuQian2020 [5]	5	42	Eric2016 [15]	Polished
5	DuQian2020 [5]	6	43	Eric2016 [15]	Shot Peened
6	DuQian2020 [5]	7	44	Gong2015 [16]	SLM-OP 1
7	DuQian2020 [5]	8	45	Gong2015 [16]	SLM-MP 2
8	DuQian2020 [5]	9	46	Gong2015 [16]	SLM-MP 3
9	DuQian2020 [5]	10	47	Gong2015 [16]	SLM-MP 4
10	Gunther2017 [6]	SLM-1a	48	Gong2015 [16]	SLM-MP 5
11	Gunther2017 [6]	SLM-1b	49	Alegre2022 [17]	As Built
12	Gunther2017 [6]	SLM-2	50	Alegre2022 [17]	HIP
13	Hu2020 [7]	noName	51	Macallister2022 [18]	AF test
14	Li2016 [8]	Edwards & Ramulu As Built	52	Mertova2018 [19]	AH
15	Li2016 [8]	Edwards & Ramulu Machined and Polished	53	Mertova2018 [19]	MH
16	Li2016 [8]	Xu et al. Annealed	54	Mertova2018 [19]	M
17	Li2016 [8]	Xu et al. AB 0°	55	Tehrani2021 [20]	AB Coarse East
18	Li2016 [8]	Xu et al. AB 90°	56	Tehrani2021 [20]	AB Fine East
19	Li2016 [8]	Hoerweder et al.	57	Tehrani2021 [20]	M Coarse East
20	Li2016 [8]	Kasperovich & Hausmann Machined and Polished	58	Tehrani2021 [20]	M Coarse West
21	Li2016 [8]	Kasperovich & Hausmann AB	59	Tehrani2021 [20]	M Fine East
22	Sanaei2020 [9]	AM250 Annealed	60	Tehrani2021 [20]	M Fine West
23	Sanaei2020 [9]	AM250 AB	61	Yan2019 [21]	AF

24	Sanaei2020 [9]	AM250 Annealed and Machined	62	Yan2019 [21]	HIP
25	Sanaei2020 [9]	M290 90° Annealed	63	Yan2019 [21]	SMAT
26	Sanaei2020 [9]	M290 45° Annealed	64	Kumar2020 [22]	3090-AF
27	Sanaei2020 [9]	M290 90° AB	65	Kumar2020 [22]	3090-HT
28	Sanaei2020 [9]	M290 45° AB	66	Kumar2020 [22]	3090-SP
29	Zhao2016 [10]	SLM-V7	67	Kumar2020 [22]	3067-AF
30	Zhao2016 [10]	H-SLM-V7	68	Kumar2020 [22]	3067-HT
31	Fousova2018 [11]	SLM	69	Kumar2020 [22]	3067-SP
32	Jiang2021 [12]	HT	70	Kumar2020 [22]	6090-AF
33	Jiang2021 [12]	LSP2	71	Kumar2020 [22]	6090-HT
34	Sun2021 [13]	Ultrasonic	72	Kumar2020 [22]	6090-SP
35	Sun2021 [13]	Rotating bending	73	Kumar2020 [22]	6067-AF
36	Gunther2018 [14]	Batch 1-1	74	Kumar2020 [22]	6067-HT
37	Gunther2018 [14]	Batch 1-2	75	Kumar2020 [22]	6067-SP

Table 9, List of all the datasets

It follows that several nested while loops are needed in order to not get lost inside the evaluation. The procedure is identical to the one carried to find the  $r$  parameter value, but this time it's fixed to  $r = 10^{-4}$ . We basically are having 76 repetitions of the training and for each subset we perform the training five times to get a 5 curves average. Also, the subset evaluation database (like the one of *Section 6.3.1*, *Table 7*) is automatically created inside the code. It may be thought that it's useless to repeat the training so many times, because in theory just 5 repetitions would be needed. But it's useful to remind that inside the custom loss function, the multidimensional space is scanned with a fixed set of features that belong to datasets that are different to evaluate. With that being said, it's trivial to see why we repeated the training so many times: each subset has its set of features and of process parameters. It is useful to confirm the expected results that we listed before: the curves of the three subsets “*Gong2015 SLM-MP 4*”, “*DuQian2020, set I*” and “*Alegre2022, As Built*” must improve with respect to the previous ones, when we removed them from the training database for the validation. The plot is simply produced like in *Figure 39*, alongside a sort of “map” that shows in the complete SN diagram, where the points of the subset are located. At the same time a **R.M.S.E.**, with respect to  $N_f$ , is computed to be used for later comparing this *P.I.N.N.* method with other ones.

Indeed, from *Figure 40*, we can see how the expectations can be fulfilled, leading to curves that are the best we have obtained so far. This means that working in a condition that is worse with respect to the one of the final evaluation, was a good strategy. However, we are not done yet. In fact, we can already see, from this preliminary insight, that the curve of the subset “*Alegre2022, As Built*” is strictly decreasing, but the concavity is not facing upward.

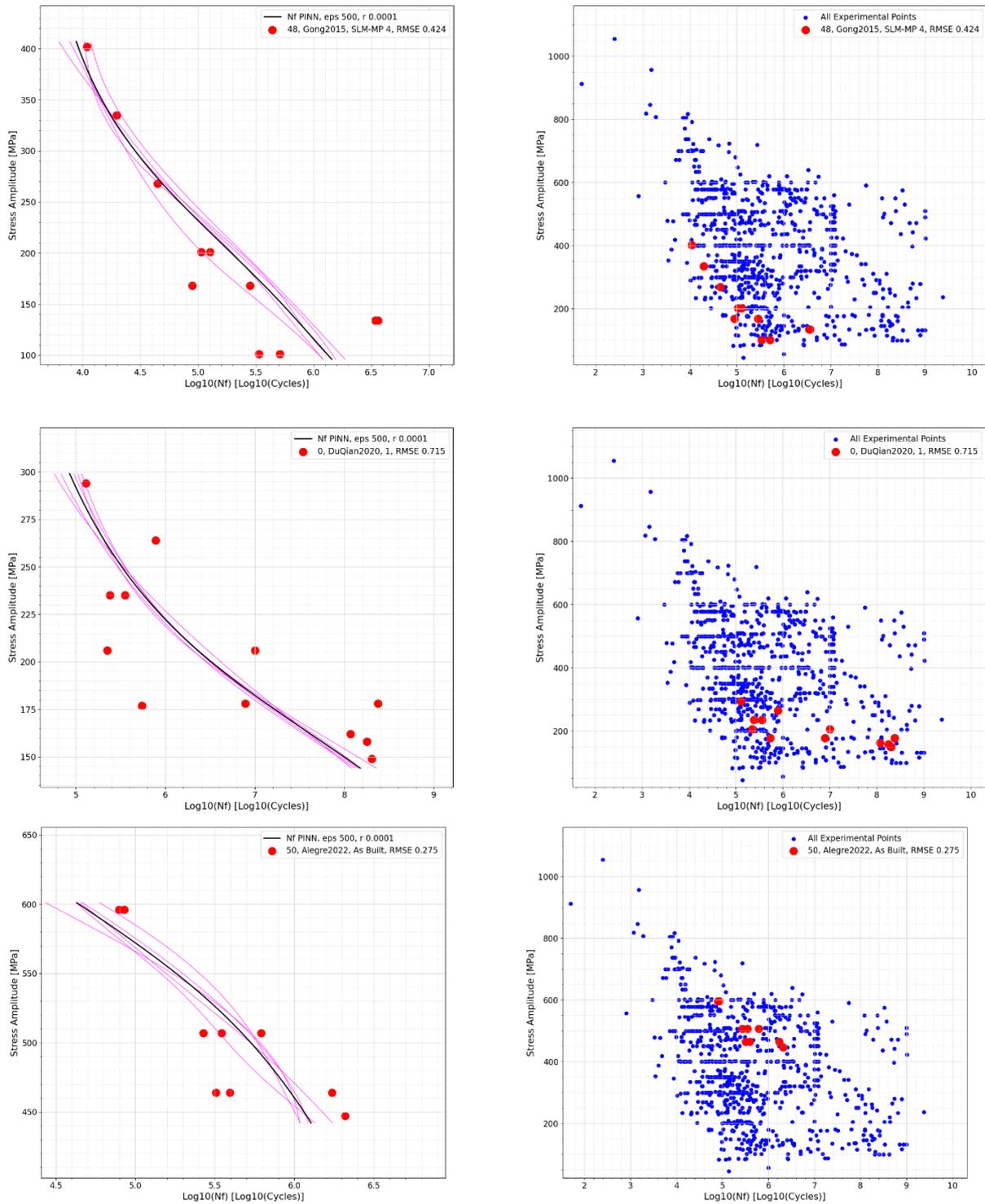


Figure 40, Fatigue curves after the tuning of the  $r$  parameter without removal from the database

This suggests that it's a useful idea to let the network understand that the second derivative must always be positive and not negative, leading, then, to an improvement of the physics informed neural network. This will be done in a later section. Running the full code, we can obtain a picture for each data set that can allow us to understand whether what we implemented is good or not and, since for each dataset the R.M.S.E. was computed (Figure 41), we can also store

separately those values to use them to compare other structures of neural network later on.

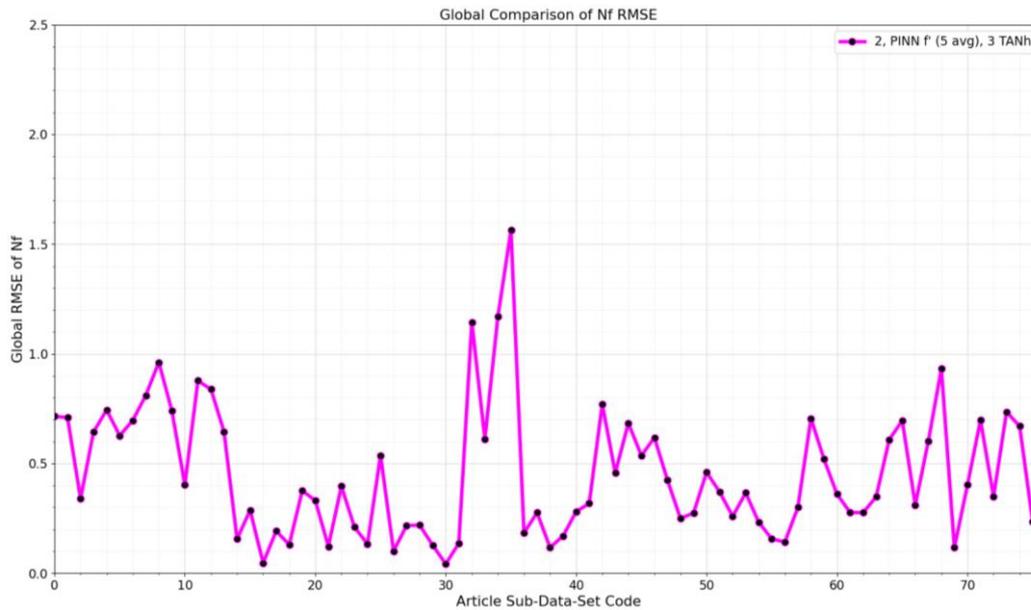
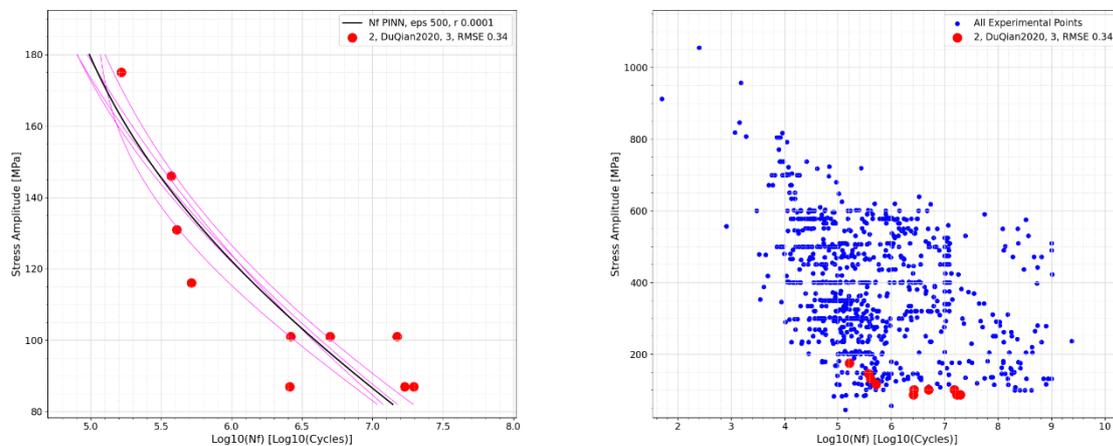
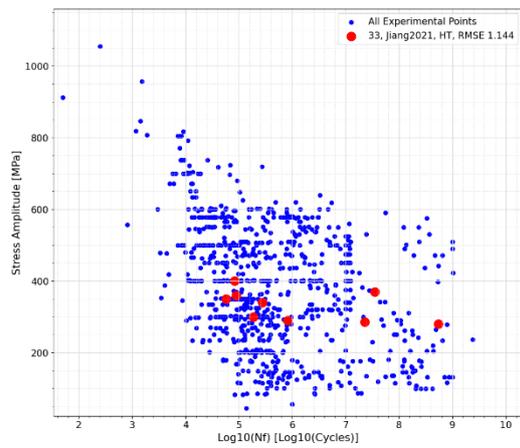
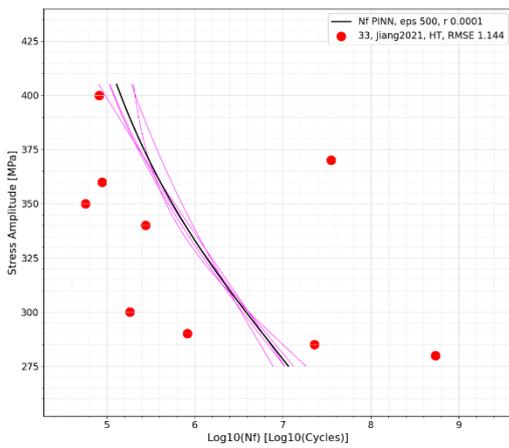
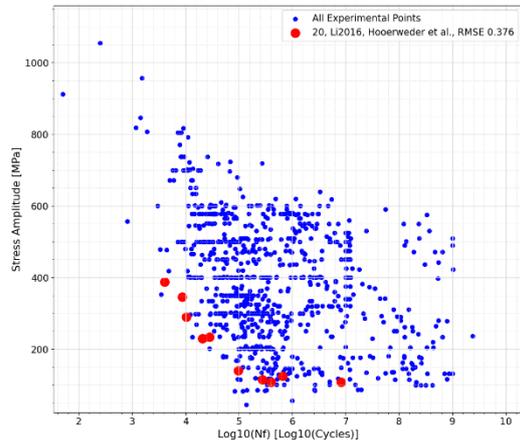
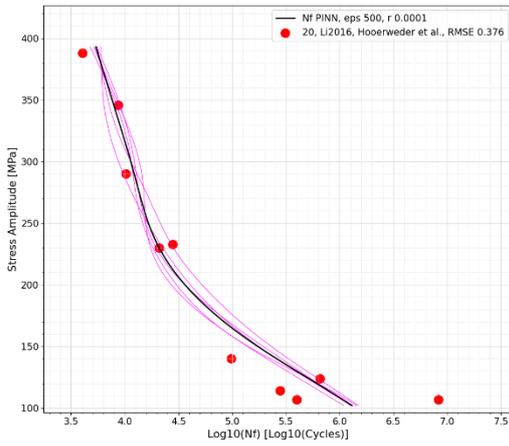
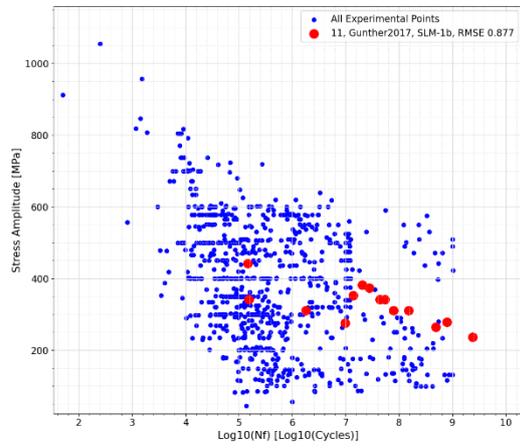
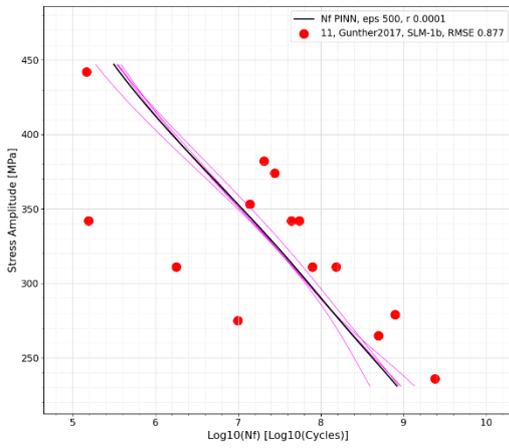


Figure 41, R.M.S.E.(Nf) evaluated for all the datasets

In fact, comparing different datasets with the same type of neural network it's not useful (like in the previously shown *Figure 41*: this was done just to see which subset has the lowest and highest R.M.S.E.), because each one of them is unrelated from the other. But comparing the same dataset R.M.S.E. with each neural network structure can be useful. It has to be reminded that, along the quantitative prediction of the R.M.S.E., it's good practice to have also a qualitative observation of the shape of the curves. Of course, showing all the pictures would be time-consuming. However, it can be stated that more than 80% of the curves have promising shapes and a good quality of the predictions (*Figure 42*).





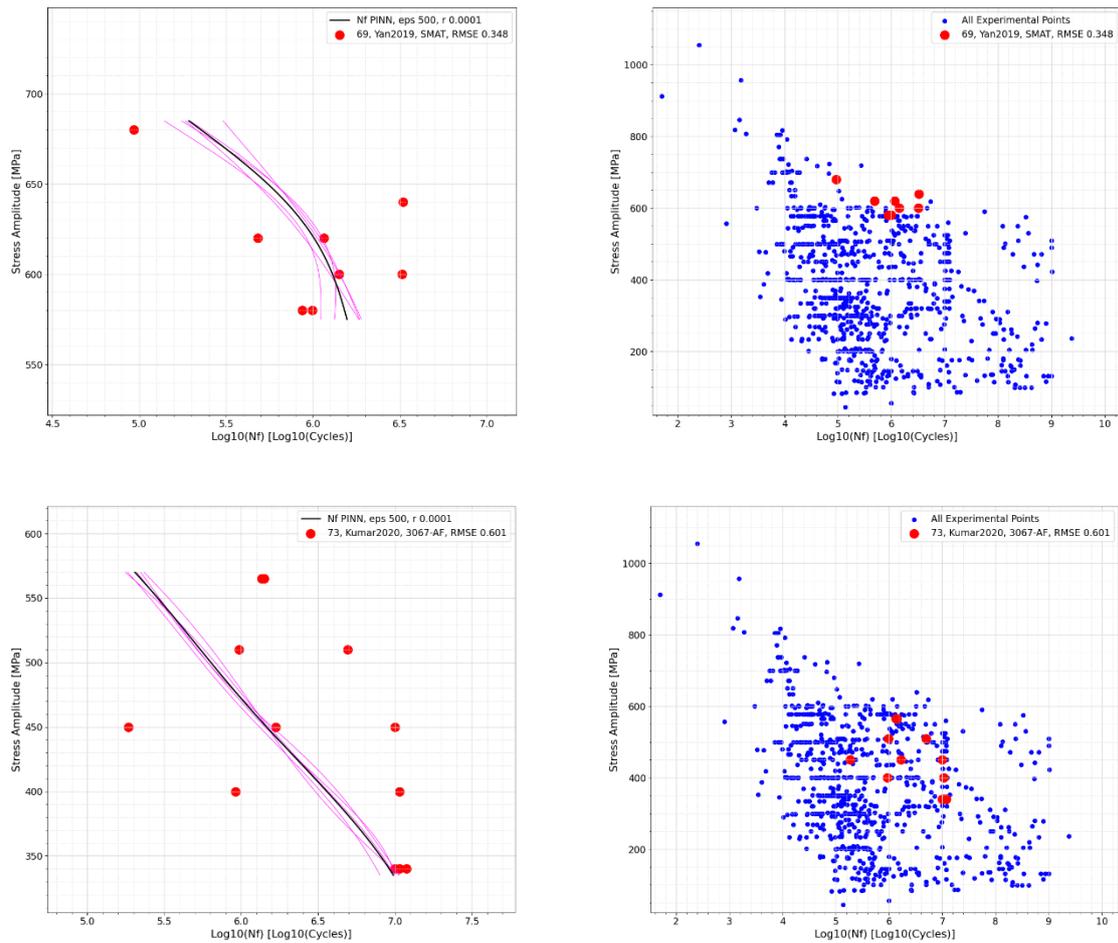


Figure 42, Datasets evaluated with the tuned Physics Informed Neural Network

Even though we have obtained a substantial improvement of the shape of the fatigue curves and of the quality of the prediction, there are still a couple of aspects to be improved:

1. There are curves that have a concavity facing downwards: we need to force the second derivative to be positive.
2. The shape of the curves is not always linear, like the one that is typical of fatigue. Even though the prediction is precise, we have several curves that are curvilinear. It would be preferable to obtain diagrams like the one shown in *Figure 11, section 3.1*.

## 7. Bilinear and concavity behavior with the P.I.N.N.

As it was stated in the previous paragraph, even though the curves have a satisfactory behavior and the predictions are almost always right, a bi-linear curve with the concavity facing upwards is preferable. To achieve this kind of behavior, we can tune accordingly, again, the structure of the neural network and, at the same time, we can implement changes to the custom loss function. It is important to remind that the objective is to get a curve that resembles the following one (*Figure 43*):

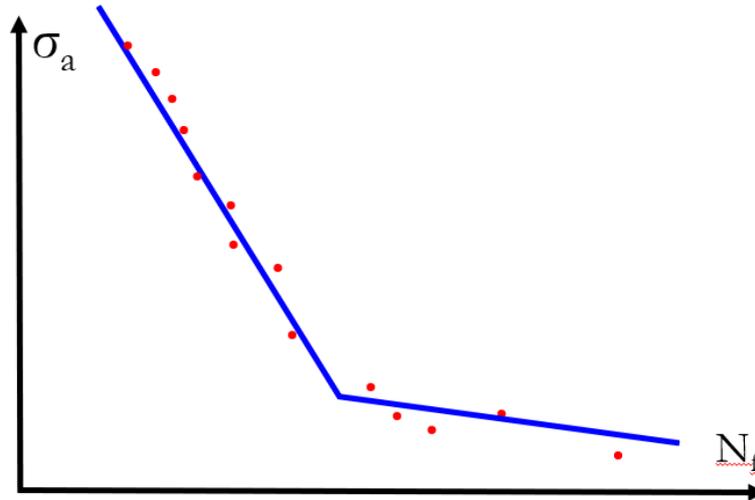


Figure 43, Typical fatigue curve

### 7.1 changing the structure of the P.I.N.N.

Recalling how *the rectified linear unit* activation function and the *linear* activation function are, it is easy to deduce that combining these ones, can give us the shape that we were looking for.

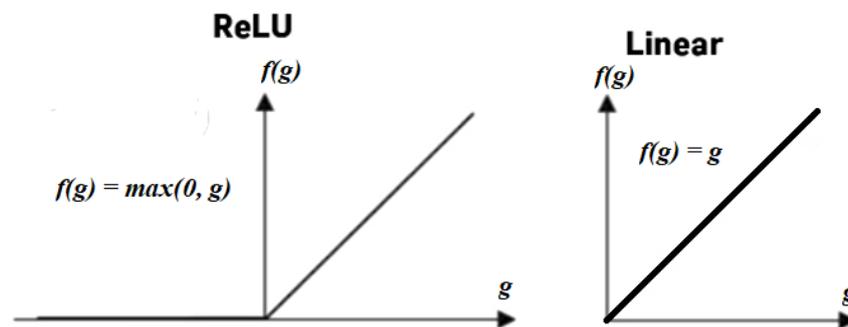


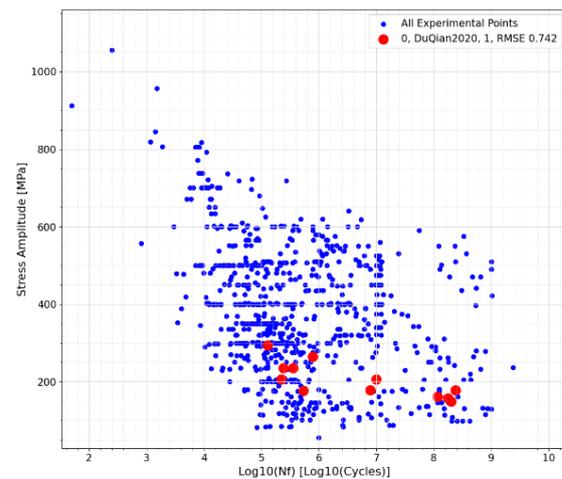
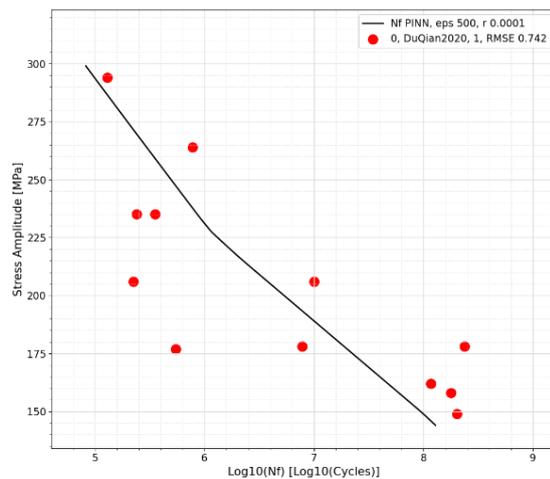
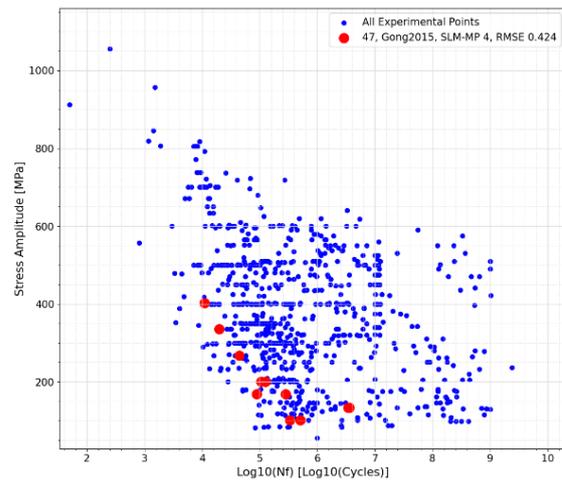
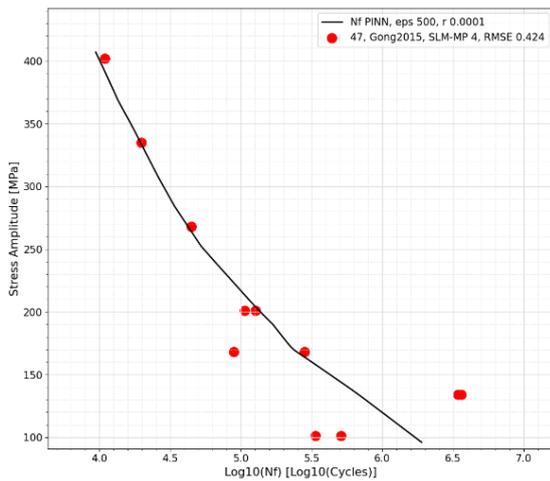
Figure 44, Activation functions used for the 3rd type of Neural Network

After extensively experimenting with several combinations, it was obtained that the best results were achieved with the structure depicted in *Table 10*. The machine learning parameters were kept equal to the previous attempts (500 Epochs, learning rate of 0.001 and batch size of 500).

Layer	N° nodes/neurons	Activation function
Input	15	-
Hidden 1	500	ReLU
Hidden 2	500	Linear
Output	1	Linear

Table 10, 3rd Neural Network structure

If we start a preliminary evaluations on the three usual sets (“*Gong2015 SLM-MP 4*”, “*DuQian2020, set 1*” and “*Alegre2022, As Built*”) we can see that the shape of the curve it's more fragmented, meaning that we have successfully reduced the curvilinear behavior, while still having a strictly decreasing trend, thanks to the implementations that were done before in the physics informed neural network algorithm with the custom loss function.



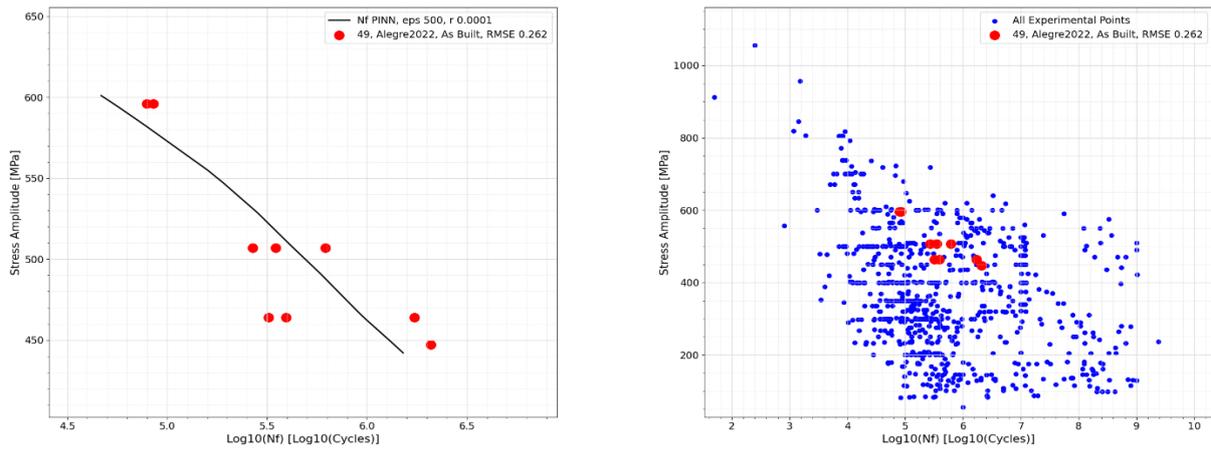


Figure 45, Evaluation of the 3 usual datasets with the 3rd Neural Network before implementing the 2<sup>nd</sup> derivative

Unfortunately, as it can be seen already from set “*Alegre2022, As Built*”, we have that the concavity is still facing downward. In fact, if we investigate other pictures, like for example “*DuQian2020, set 7*” (Figure 46), we can see that we need something more accurate to solve this problem. Because, with respect to the previous curvilinear attempt, the situation is even worse.

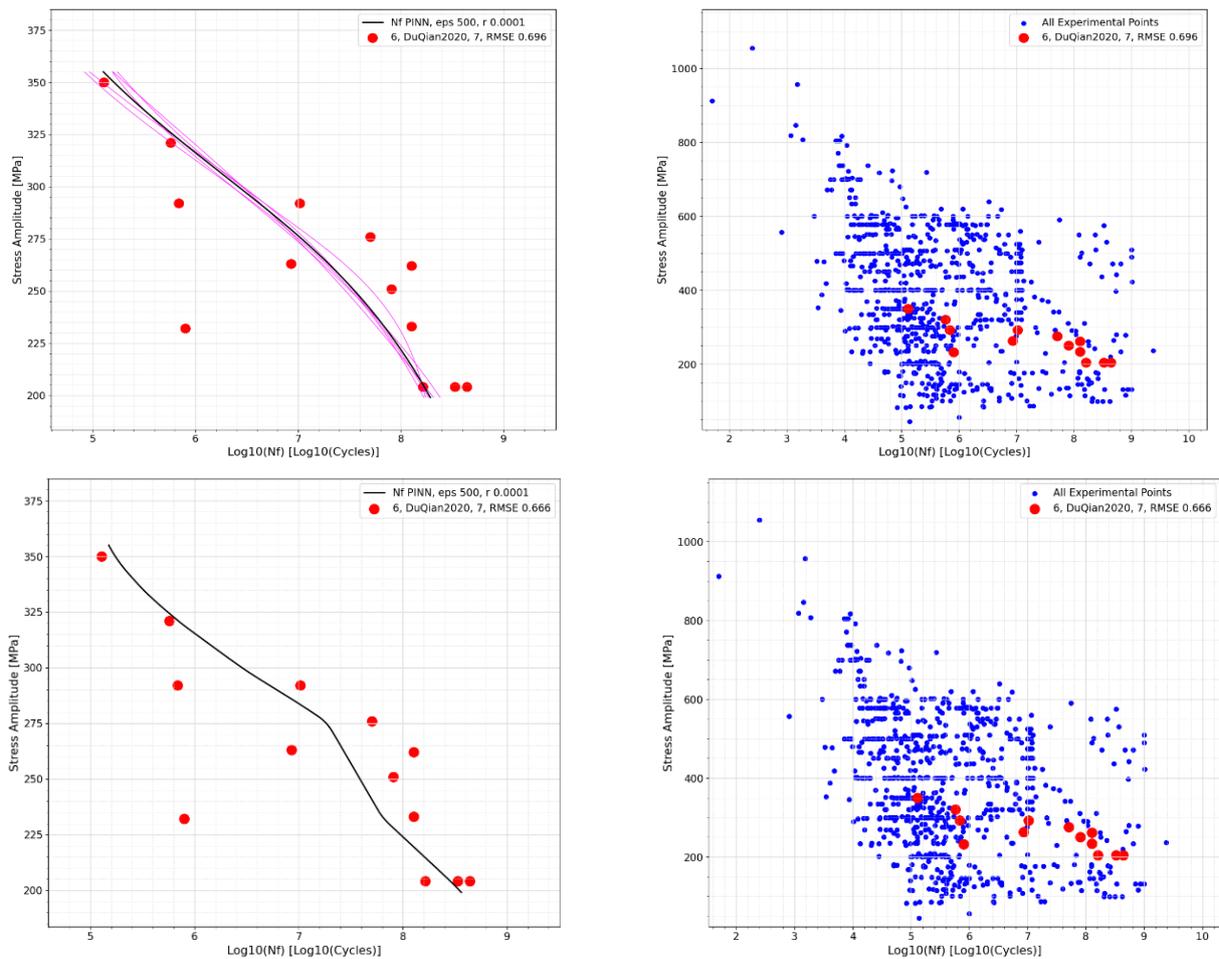


Figure 46, Fatigue curves with the concavity facing upwards with the 2<sup>nd</sup> and 3<sup>rd</sup> Neural Network type

## 7.2 Considering the 2<sup>nd</sup> derivative in the custom loss function

To make sure that we have a concavity always facing upwards, it's necessary to introduce a new strategy inside the custom loss function. This strategy is very similar to the one that has already been discussed in *Section 6.3*, but, at the same time, we have also to make sure that now, a strictly decreasing behavior can be achieved. To obtain this, we must have a *custom penalization function*  $f$  that takes into account both the first derivative and the second one. In a similar fashion to what has been already implemented in the previous section, we still have a number of sampling points, that are used to calculate the first derivative in the SN plane, with all the other features (process parameters and treatments) kept constant, except the stress amplitude, which varies in the range of the article subset that we are interested in evaluating. Formally speaking, we are computing a gradient in a space with 16 dimensions, but 14 of them are constant, so it's like being inside a plane, in which the stress amplitude  $\sigma_a$  is used to drive the prediction of the number of cycles  $N_f$ . It's important to remember that all the variables involved are *TensorFlow* variables, more precisely, they are tensors, because this is what is used inside the *TensorFlow* library to make computations. Once we have obtained the first derivative  $dN_f/d\sigma_a$ , we can make the second one, by differentiating the first one with respect to the stress amplitude.  $d^2N_f/d\sigma_a^2$  is computed in the same exact way as the first derivative: the space is sampled with the sampling points with all the features kept constant, except the stress amplitude, which is used to differentiate the freshly computed first derivative. Once we have both the first and second derivative, we can create a newly updated custom penalization function  $f$ , that allows the neural network to give the curves the desired shape. This latter one is based off the one that has been implemented inside *Section 6.3*. The function is the following one:

$$f = \frac{-\frac{dN_f}{d\sigma_a} + \left| \frac{dN_f}{d\sigma_a} \right|}{2} + c \cdot \frac{\frac{d^2N_f}{d\sigma_a^2} + \left| \frac{d^2N_f}{d\sigma_a^2} \right|}{2} \quad \text{with} \quad g = \frac{dN_f}{d\sigma_a}; \quad h = \frac{d^2N_f}{d\sigma_a^2}$$

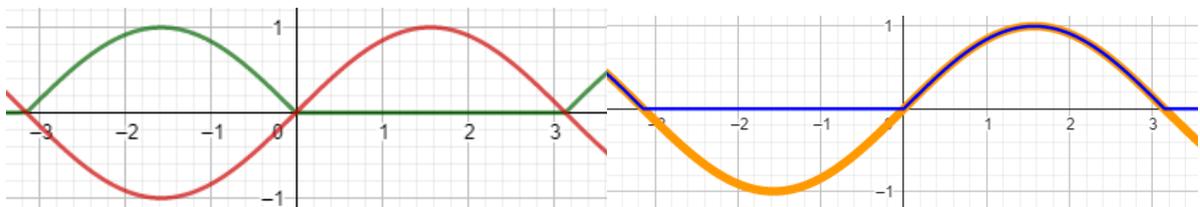


Figure 47, customized penalization function to take into account the 1st and 2nd derivative

The discussion of the green curve is the same as the previous chapter: since we want to penalize positive derivatives, we give a positive weight to the negative derivatives, while we give a null weight to the positive derivatives. For the blue

curve we have similarly fashioned reasoning but reversed. If we assume that the second derivative (*orange*) has the depicted trend (in this case depicted as a sinusoidal for the sake of simplicity, but, again, it can have any kind of shape), since we want to penalize negative derivatives, we are giving a positive weight to the positive second derivative (which corresponds to a concavity facing upwards, that is the desired behavior) while we are giving a null contribution to the negative derivative. This is represented by the expression and the curve in *blue*. The *green* (representing the behavior of the first derivative) and the *blue* (referred to the second derivative) expressions are combined through a sum. Then the function  $f$  is fed inside the custom loss function in the same exact way that has been done before. The expression that dictates the behavior of the second derivative has a weight marked by the letter  $C$ . This last one can be tuned accordingly with a procedure that is very similar to the one adopted for the  $r$  parameter. A vector of  $C$  ranging from  $10^{-12}$  to  $10^{+12}$  has been created, and 10 subsets were evaluated per each value of  $C$ . The 10 curves of each value of  $C$  were carefully analyzed and at the end of this qualitative evaluation, the value of  $10^5$  was chosen. This value worked well because it allowed to reduce the concavity facing downward.

In fact, after implementing those changes, that take into account the behavior of the second derivative inside the custom loss function, we can see that we have a significant improvement on the behavior of the concavity. If we take as example the set “*DuQian2020, set 7*” We can see that there has been a correct penalization of the negative second derivative.

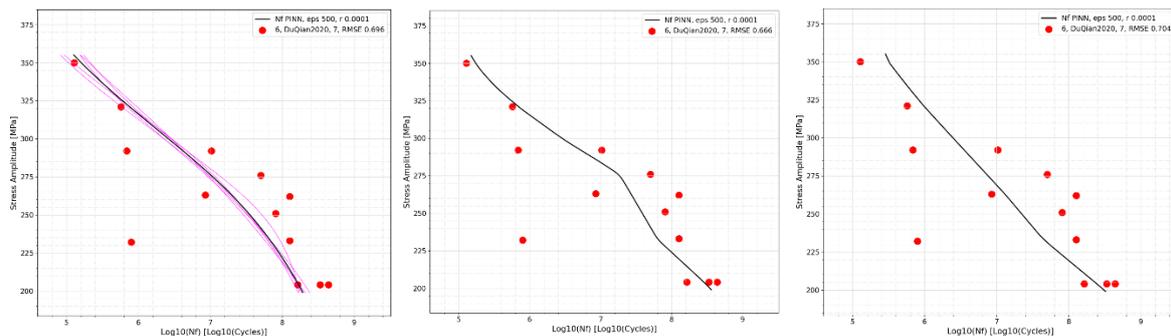


Figure 48, Improvement of the concavity behaviour for the set DuQian2020, 7

The same upgrade can be observed for the afore mentioned set, “*Alegre2022, as built*”:

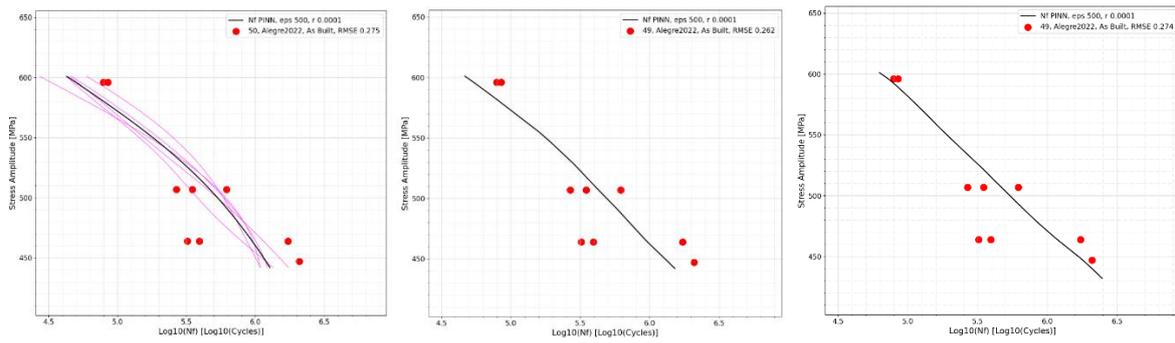
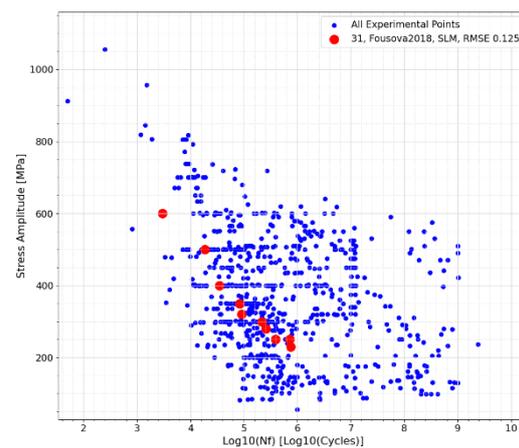
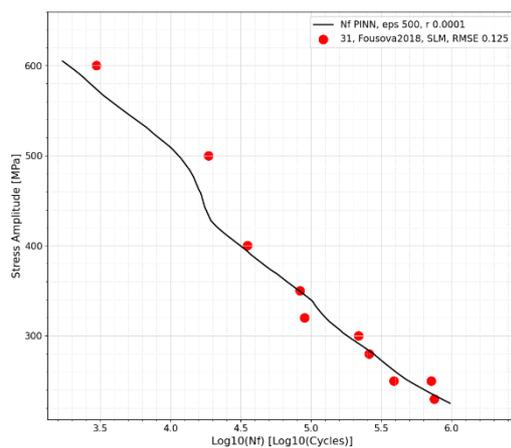
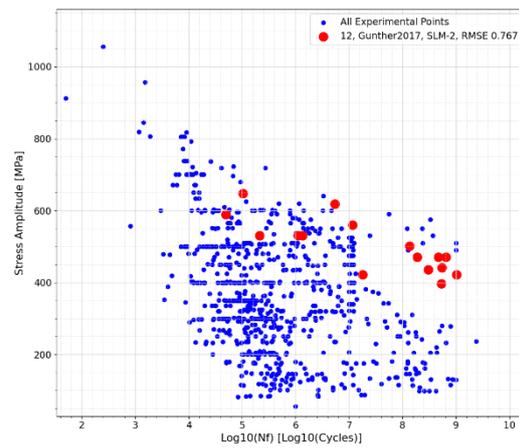
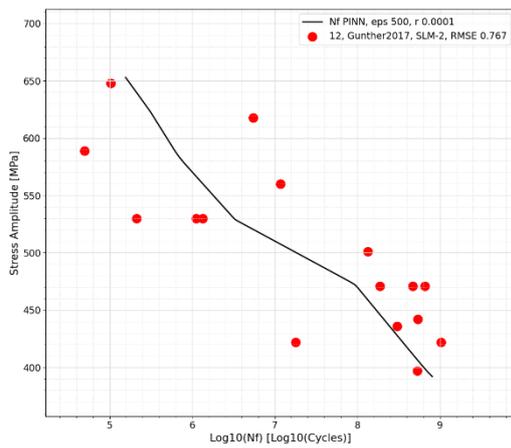


Figure 49, Improvement of the concavity behaviour for the set Alegre2022, As Built

However, after producing a curve for all the subsets of the database, it was observed that there is still a small number of sets that has the concavity facing downwards. Here below are reported a couple of them.



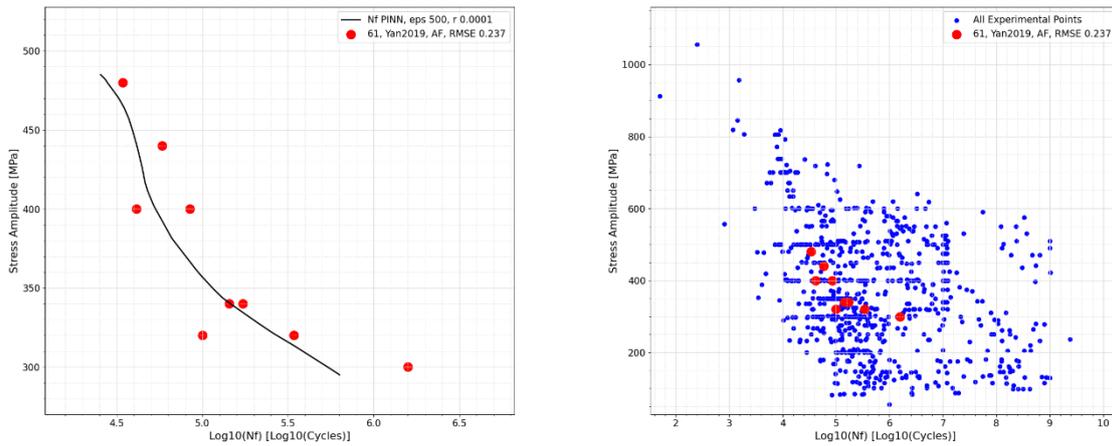


Figure 50, Curves, that after penalizing negative 2<sup>nd</sup> derivatives, still have the concavity facing downwards

Those pictures suggest that is necessary to investigate another method to get better predictions and to mitigate the wrong behaviors once and for all. Because even though their predictions are improved, we are not always getting curves that are typical of fatigue testing.

## 8. F.F.N.N. with concatenated layers

Another attempt that was tried is a more straightforward and simpler one. It was observed that, in order to complete the training and evaluation of the previous types of neural networks (the five-curve averaged one and the ReLU- Linear one), it was required an extensive amount of time. The training requires even more time when we are evaluating the *physics informed neural network* with a custom loss function, with respect to a simple *feedforward neural network*. So, it was thought to use the "Mean Squared Error" loss function, that is already integrated inside the machine learning environment, leading to have a simple *feedforward neural network* as the one that was used in Section 5. But this time, along with the power-speed feature cross and an enriched database, it was decided to use a new structure for the neural network. Having kept the machine learning parameters as the usual ones (500 epochs, learning rate = 0.001 and batch size = 500), the structure this time is composed by one single layer that is the result of the concatenation of a **ReLU** and **linear** layer, both made of 1000 neurons.

To implement this, we needed to sequentially create the input layer, the linear and ReLU layer, that are not placed in series but in parallel, both taking as input the input layer, and, after performing their concatenation via the command `".layers.Concatenate"`, the concatenated layer is sent to the usual single node linearly activated output layer. This concatenation allowed faster training times. *Figure 51* resumes what has been just explained:

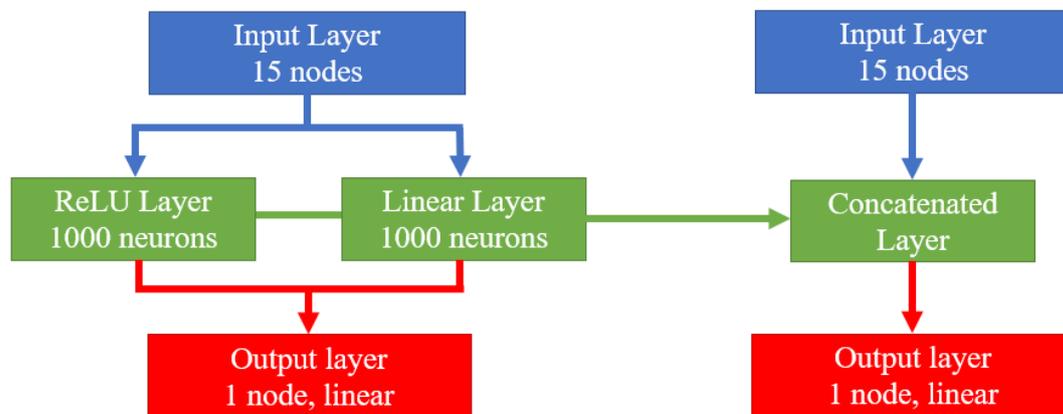


Figure 51, Logical scheme for the concatenated feed forward neural network

Then it's possible to evaluate all the single articles as it has been done in all the previous sections. The root mean squared error is also computed for each article in the same exact way as the previous methods. The idea is to compare the R.M.S.E. of all the methods. It is first useful to see what happens to the usual three datasets that we have been evaluating preliminarily in all the sections: "Gong2015 SLM-MP 4", "DuQian2020, set 1" and "Alegre2022, As Built".

Indeed, we can see that this strategy allows us to get good results without having the complication of the physics informed neural network.

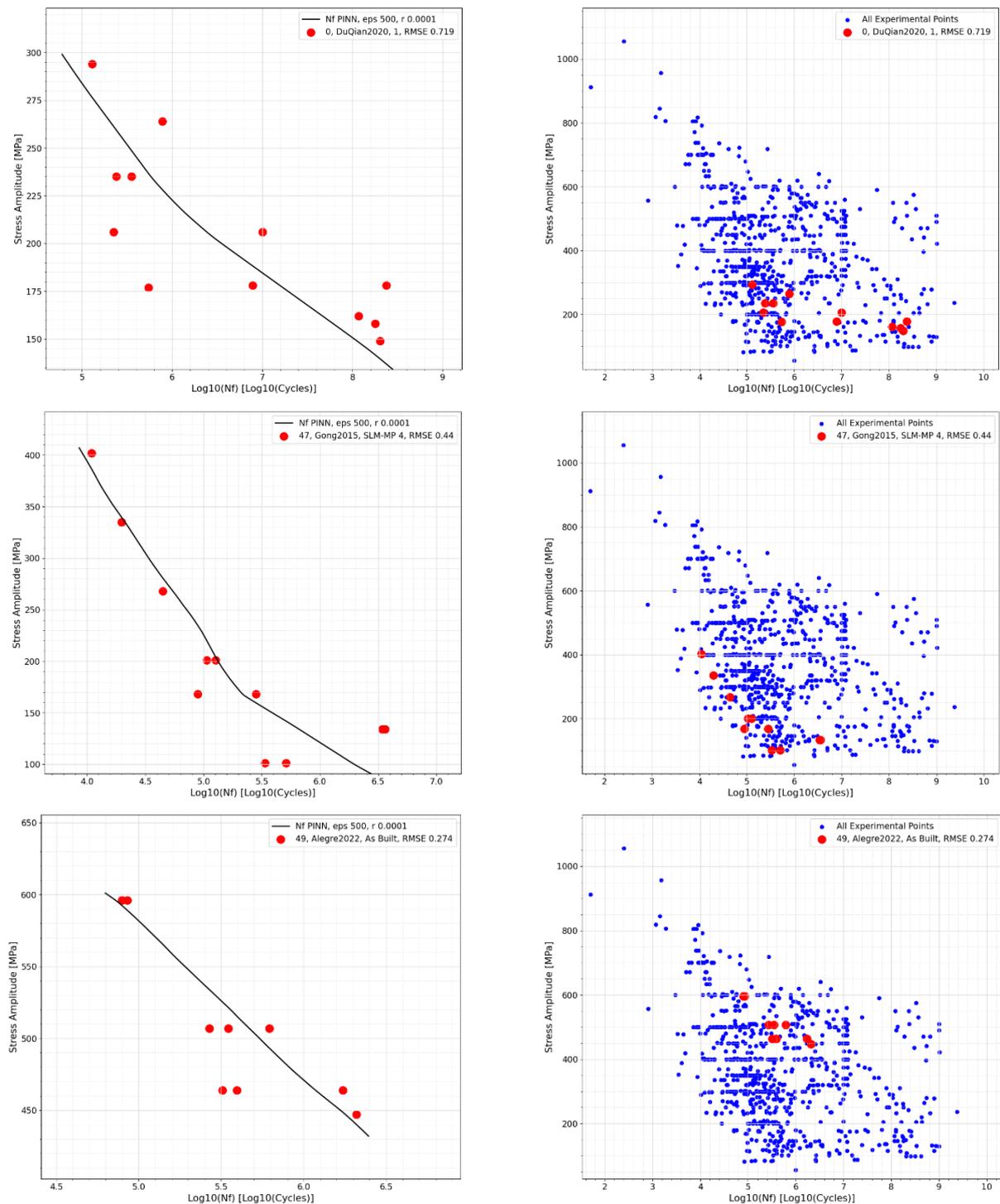
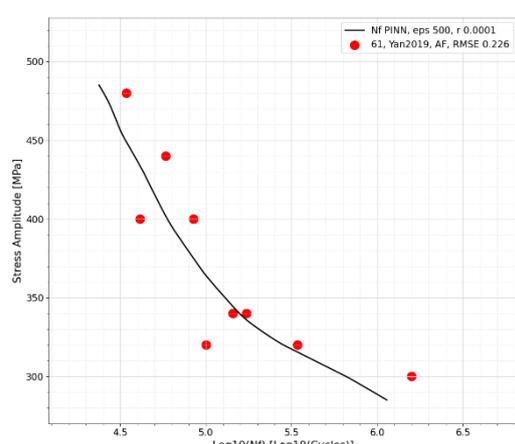
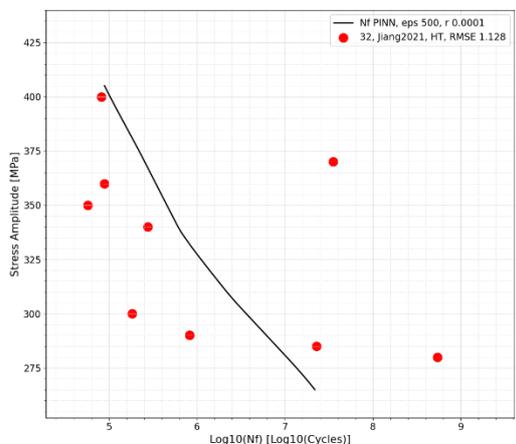
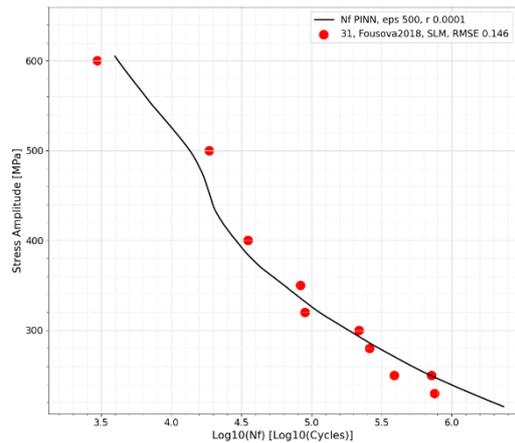
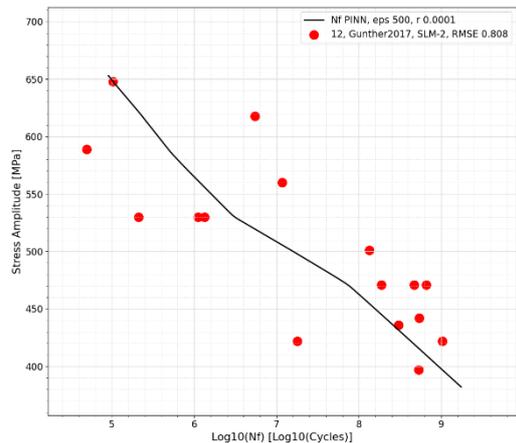
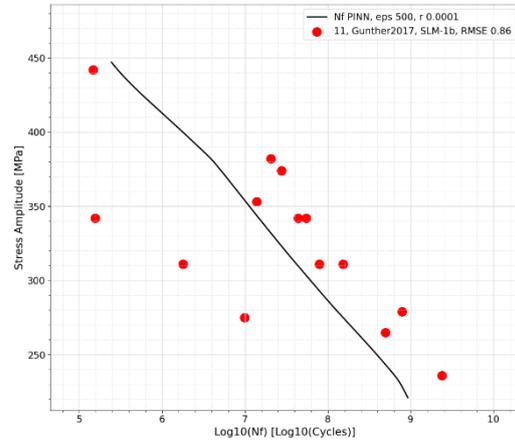
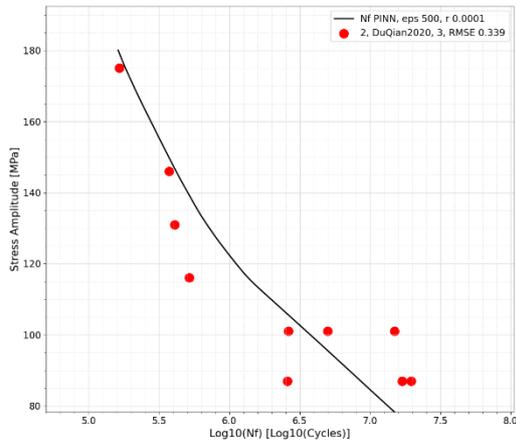


Figure 52, Three usual datasets evaluated with the 4th neural network type: Concatenated F.F.N.N.

In fact, we can see that we get curves that are nearly identical to the typical shape of fatigue problems. They are almost strictly decreasing with a seemingly almost linear behavior. However, if we investigate the same article subsets that have

been shown in *Section 6* and *7* with their figures, we can see the drawbacks of this method: we still have the concavity facing downwards in some datasets and in just a few, we have small sections that are not strictly decreasing.



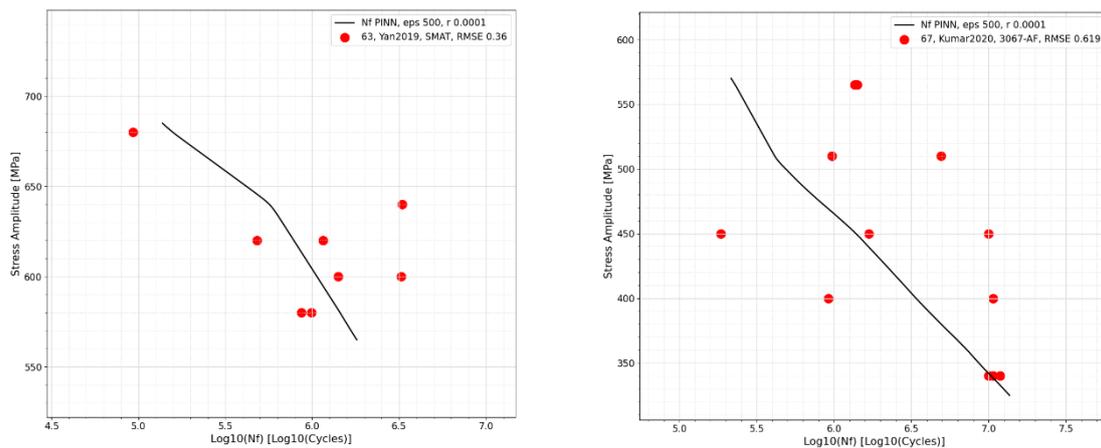


Figure 53, Datasets evaluated with the 4th type of neural network: concatenated F.F.N.N.

Although it has been surprisingly pleasurable to see that such a small change in the structure of the neural network with just a simple F.F.N.N. allowed to get results that are similarly good to the ones of the complicated P.I.N.N., we are not still in the condition of obtaining bilinear curves. We could have used just a single *linear* hidden layer to easily get a single line in the fatigue diagram, but it is preferable to have a bi-linear fit.

## 9. Pure Bi-Linear Behavior

Up until now we have been using these four types of strategies in order to get the desired predictions:

1. Feed Forward Neural Network with 3 ReLU layers
2. Physics Informed Neural Network with positive 1<sup>st</sup> derivative penalization and 3 tanh layers
3. Physics Informed Neural Network with positive 1<sup>st</sup> derivative penalization, negative 2<sup>nd</sup> derivative penalization and ReLU-Linear Layers
4. Feed Forward Neural Network with ReLU-Linear concatenated layer

At the end it was observed a progressive improvement, as *Figure 54* shows on the set “*Gong2015 SLM-MP 4*”, but the pure bi-linear behavior was never actually achieved.

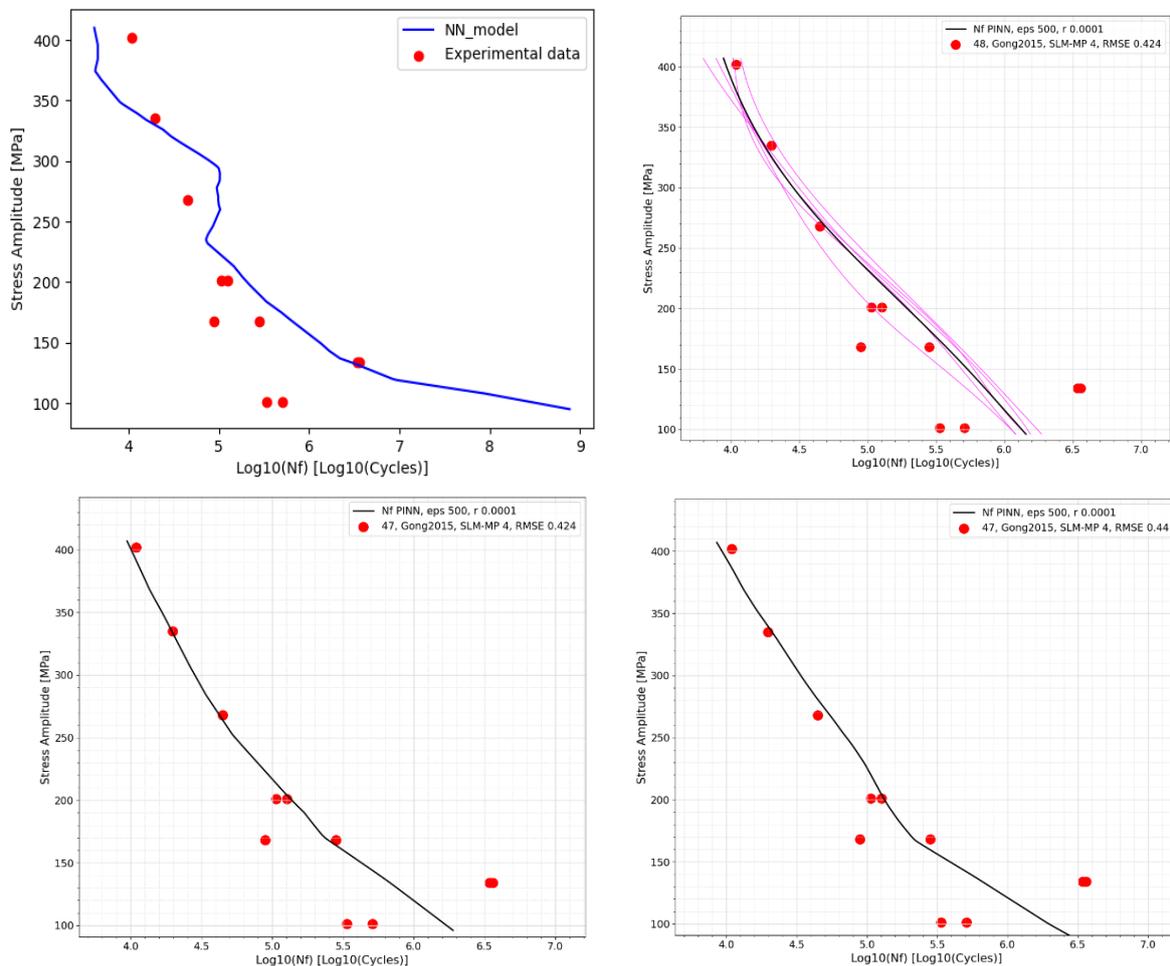


Figure 54, History of the 4 neural network types seen up to now

We need a completely new approach and method to achieve a pure bi-linear behavior. The database must be rethought in terms of dimensions and machine learning features and labels. To this end it is important to remind how a simple bilinear function is built.

## 9.1 The Bi-Linear Function

A bilinear function is simply made of two lines that are intersecting in a point (**Point \***). These latter ones have a slope ( $m_1$  and  $m_2$ ), that is basically showing us how the line is inclined, and a bias ( $q_1$  and  $q_2$ ), that tells us where the lines are intersecting the Y axis. For our case, since we are in a SN plane ( $\sigma_a$  and  $S$  are interchangeable and the same holds for  $N_f$  and  $N$ ), the graphical situation is explained by *Figure 55*.

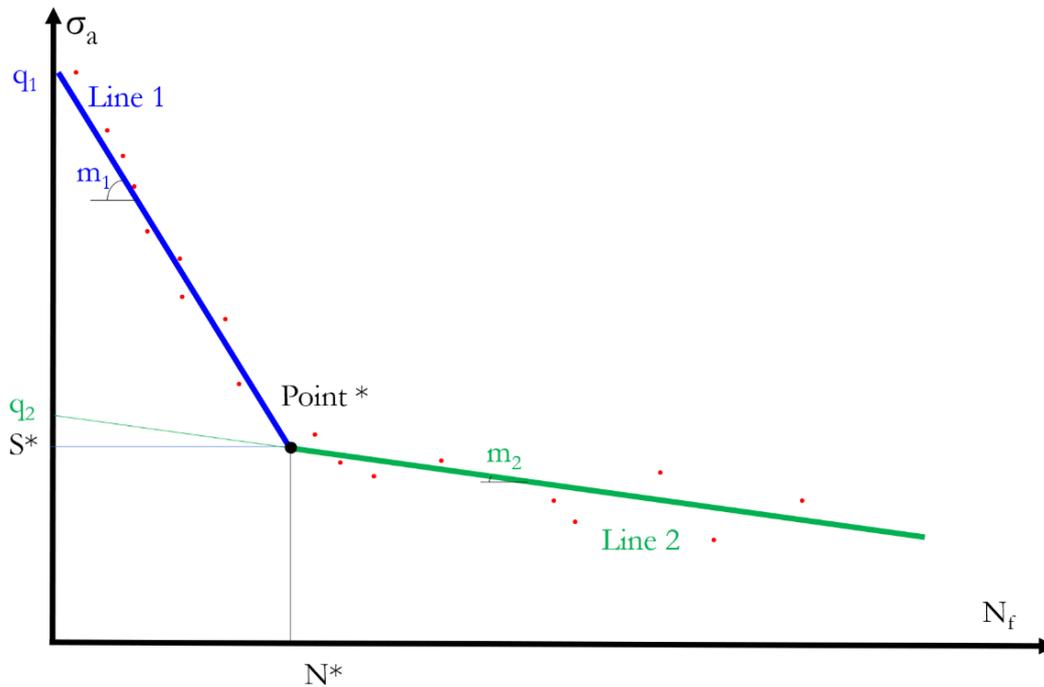


Figure 55, Mathematical layout for the Bi-Linear approach

The equations of the problem are the following ones:

$$\begin{cases} S = m_1 \cdot N + q_1 \\ S = m_2 \cdot N + q_2 \end{cases}$$

However, the bilinear function is piecewise defined, meaning that we have to tell it when we are considering the first equation or the second one. To obtain this, it's necessary to find the intersection **point \***. If we simply replace  $S$  with  $S^*$  and  $N$  with  $N^*$  we obtain the following linear system:

$$\begin{cases} S_* = m_1 \cdot N_* + q_1 \\ S_* = m_2 \cdot N_* + q_2 \end{cases}$$

By performing trivial mathematical calculations, we can obtain the expression for the coordinates of the **point\***:

$$N_* = \frac{q_2 - q_1}{m_1 - m_2}; \quad S_* = m_1 \cdot N_* + q_1$$

With the *point* \* found, we can finally tell Python when to consider one expression or the other:

$$\begin{cases} S = m_1 \cdot N + q_1 & \text{if } N \leq N_* \\ S = m_2 \cdot N + q_2 & \text{if } N > N_* \end{cases}$$

Since we have been interpreting this machine learning problem and the fatigue graphs in the opposite way, so using  $S$  as an independent variable and  $N$  as a dependent one, still plotting, however, in the correct way, we can also find the expressions of  $N$  as a function of  $S$ . Because when plotting the lines, we are first looking for the stress amplitude  $S$  range depending on the selected article, and only then we are determining  $N$ . By inverting the expressions, it holds that:

$$\begin{cases} N = \frac{S - q_1}{m_1} & \text{if } S \geq S_* \\ N = \frac{S - q_2}{m_2} & \text{if } S < S_* \end{cases}$$

## 9.2 How the Bi-Linear database is built

The idea that lies behind this section is to build a database that is nearly identical to the one that has been used up until now. The only difference being that, before we were predicting the number of cycles directly, now the idea is to predict the 4 bilinear parameters  $m_1$ ,  $m_2$ ,  $q_1$  and  $q_2$ . So, we have 15 features (process parameters, thermal and surface treatments) and 4 labels (the 4 bilinear parameters). *Table 11* illustrates what has just been told:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
1																							
2																							
...																							
i																							
...																							
76																							

A	Article Subset Name	L	Machined Boolean
B	Article Name	M	Sand Blasted Boolean
C	Subset Global Code	N	E.D.M. Boolean
D	Build Orientation [°]	O	L.S.P. Boolean
E	Power [W]	P	S.P. Boolean
F	Scan Speed [mm/s]	Q	S.M.A.T. Boolean
G	Hatch Distance [mm]	R	Surface Polishing Boolean
H	Layer Thickness [µm]	S	Parameter $m_1$
I	Annealed Boolean	T	Parameter $m_2$

J	Annealing Temperature [°C]	U	Parameter $q_1$
K	H.I.P. Boolean	V	Parameter $q_2$

Table 11, Training database for the Bi-Linear approach

This time the feature cross was not used, as it was referred only to the previous types of problem, and this one is a completely different one in terms of machine learning. In this case, instead of having several hundred rows, we are just having one row per each article sub-data-set. In total we have just 76 rows. This is also beneficial for the machine learning algorithm, as this low numerosity lightens the calculations. The only task that we are left with is populating the columns that are referred to the 4 bilinear parameters.

### 9.2.1 Finding the 4 bilinear parameters

In order to find the 4 bilinear parameters, a specific code with several nested while loops, has been built. Its task is to choose two points: one in the top left region of the graph (**Point A** with coordinates  $(N_A; S_A)$ ) and the other one in the bottom right part (**Point B** with coordinates  $(N_B; S_B)$ ). These two points are chosen specifically for each data set, with respect to how these regions are populated. Then for each of these two points, a set of straight lines passing from one point, is produced by using the following trivial equations:

$$\begin{cases} (S - S_A) = m_1(N - N_A) \\ (S - S_B) = m_2(N - N_B) \end{cases}$$

The only thing that is left to be chosen are the slope coefficients  $m_1$  and  $m_2$ . To make a clever selection, for each dataset a range of  $m_1$  and  $m_2$  was imposed. For  $m_1$  the lowest possible value starts with a high negative number and ends at  $m^*$  (which is still negative), while  $m_2$  ranges from  $m^*$  to a very low negative number.  $m^*$  is the slope coefficient of the line passing from point A to point B (having a negative sign with a relatively low absolute value), while the other two values are chosen accordingly and tuned to the considered set of points. As an example, we might have for a data set that the vectors of  $m_1$  and  $m_2$  are ranging in the depicted manner:

$$\begin{aligned} m_1 &= [-1000, -999, \dots, m^*] \\ m_2 &= [m^*, \dots, -0.002, -0.001] \end{aligned}$$

The step inside the vector is imposed manually considering that single data set, so the one that is reported for this example is not representative of all the cases. By imposing the negative coefficients  $m_1$  and  $m_2$  in this manner, we are also injecting the physics of the problem having always slopes that are negative and the concavity that is facing upwards. *Figure 56* shows conceptually what has been

explained. But how do we choose which is the best  $m_1$  and  $m_2$ ? Of all the possible combinations that are available, which one is the best in terms of fitting?

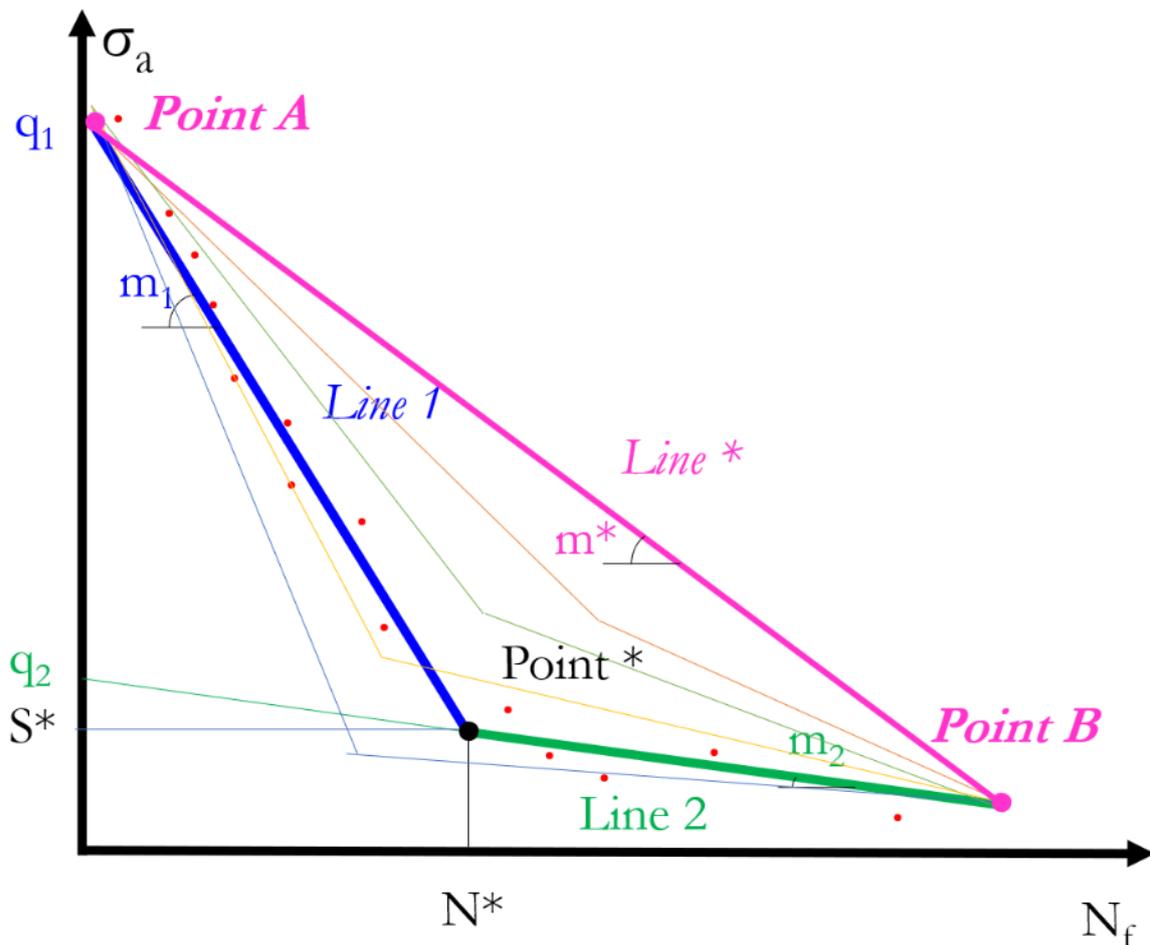


Figure 56, Logical layout to find the best combination of the 4 Bi-Linear parameters

Of all the possible combinations of  $m_1$  and  $m_2$ , it is chosen the one that produces the lowest *root mean square error* with respect to the number of cycles  $N_f$ . So, the code that calculates the R.M.S.E. can be employed also for this purpose. The vectors of  $m_1$  and  $m_2$  are scanned creating all the possible combinations for the chosen ranges. Then, for each combination, the expression of the bilinear function is produced, considering that, once the points A and B are decided and the combination of  $m_1$  and  $m_2$  analyzed is determined,  $q_1$  and  $q_2$  are automatically found through the following equations:

$$\begin{cases} q_1 = S_A - m_1 N_A \\ q_2 = S_B - m_2 N_B \end{cases}$$

With  $m_1$  and  $m_2$  found, we can compute *point \** and get the expression of the bilinear function, from which we are going to calculate the distances to the experimental points to get the R.M.S.E.. Then the *root mean square error* value is stored into a matrix that has the same dimensions of the vectors of  $m_1$  and  $m_2$ .

If, for example, the vector of  $m_1$  has 200 items and vector  $m_2$  has 150 elements, there will be  $200 \cdot 150 = 30\,000$  iterations and 30 000 R.M.S.E., as the matrix will have size  $200 \times 150$  and 30 000 elements. By performing the minimum of this matrix, we can finally find the most suitable value for the analyzed article sub-data-set. It's imperative that the starting value of the  $m_1$  vector and the ending value of vector  $m_2$ , are chosen accordingly, to avoid creating lines that are completely out of the range of the experimental points of the considered data set. To this end, the bilinear function that derives from the chosen value of  $m_1$  and  $m_2$ , is plotted to verify that the result is the desired one.

This process is repeated for all the 76 sub-data-set, allowing us to finally fully populate the training database for this new method.

### 9.3 Tuning the bi-linear network machine learning parameters

Since the training database has a new structure with respect to the one that has been used in the previous attempts, it's a good practice to repeat the tuning of the neural network parameters. After testing several configurations, it was found that the following structure of neural network was working best:

Input Layer	15 Nodes
Layer 1	100 Neurons (tanh)
Layer 2	75 Neurons (tanh)
Layer 3	50 Neurons (tanh)
Output Layer	4 Nodes (Linear)

Learning Rate	0.001
Epochs	1000 – 10 000
Batch Size	40

Table 12, Structure and Hyper-parameters of the Bi-Linear F.F.N.N.

In this case, unlike in the previous attempts, the activation function does not affect the shape of the final curve, as the bi-linear behavior is forced with the four coefficients. Since the size of the database is much smaller with respect to the previous attempts, the training proceeds much faster, therefore we can also increase the number of epochs to have a more accurate prediction. In fact, since the bilinear behavior is forced with the four coefficients, increasing the number of epochs will not create an overfitting behavior as it happened in *Figure 24*. In addition, we are not using any custom loss function, so the type of neural network is a simple F.F.N.N., allowing us to train the model with lighter calculations. As if it wasn't enough, another important advantage of this type of model is that the training can be just performed once for all the datasets and the articles are

evaluated later with a while loop. One last concept to remember is that it's absolutely necessary to put 4 nodes in the output layer, because with this neural network we are predicting 4 labels and not one, as in the previous case. Because of this, the neural network outputs four normalized results that have to be denormalized separately. In fact, as it was explained in *Section 5*, before feeding the network with the training database, it has to be normalized. For this matter, also the four labels are normalized before the training with their mean values and standard deviations. The neural network produces as output a 1x4 vector, after that we feed the command *model.predict* with the correct sets of process parameters pertaining to the subset that we are evaluating.

	0	1	2	3
<b>N.N. normalized prediction</b>	$m_{1, NN, norm}$	$m_{2, NN, norm}$	$q_{1, NN, norm}$	$q_{2, NN, norm}$

$$\begin{cases} m_{1, NN} = m_{1, NN, norm} \cdot m_{1, Std.Dev.} + m_{1, mean} \\ m_{2, NN} = m_{2, NN, norm} \cdot m_{2, Std.Dev.} + m_{2, mean} \\ q_{1, NN} = q_{1, NN, norm} \cdot q_{1, Std.Dev.} + q_{1, mean} \\ q_{2, NN} = q_{2, NN, norm} \cdot q_{2, Std.Dev.} + q_{2, mean} \end{cases}$$

Once the four coefficients are denormalized, we can implement the usual bilinear expression and the plot of the curve by calculating also the root mean square error.

## 9.4 Results

If we proceed with the evaluation of all the datasets, we can finally affirm that the objective, that has been pursued in the previous sections, has been finally achieved:

1. The curves are bi-linear,
2. A strictly decreasing behavior is observed,
3. The concavity is always facing upwards,
4. The quality of the prediction is good.

Here below are reported the three usual data sets that have been analyzed up to now: “*Gong2015 SLM-MP 4*”, “*DuQian2020 set I*” and “*Alegre2022, As Built*”. In *green* we have the bi-linear fit, while in *orange* the predicted curve.

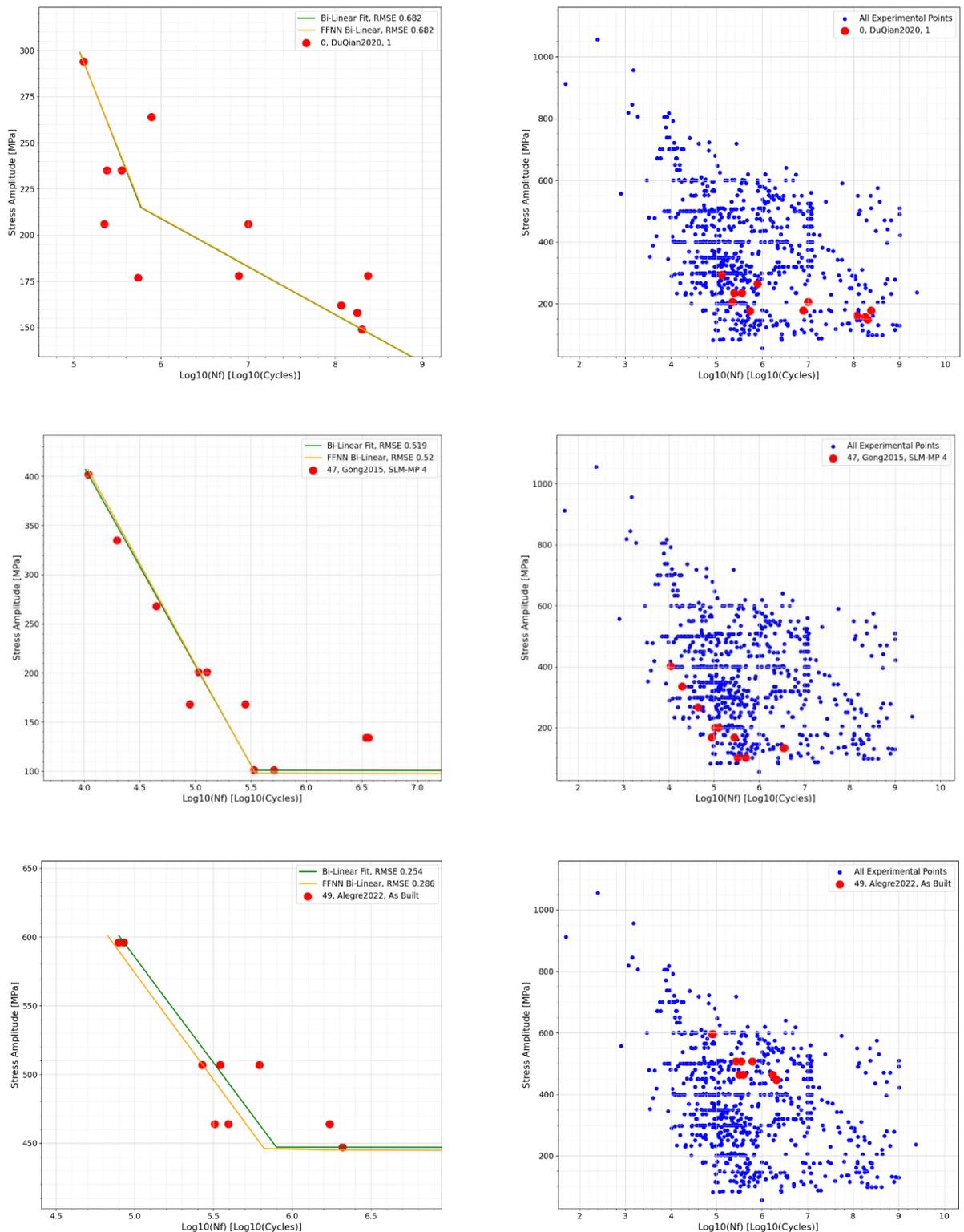
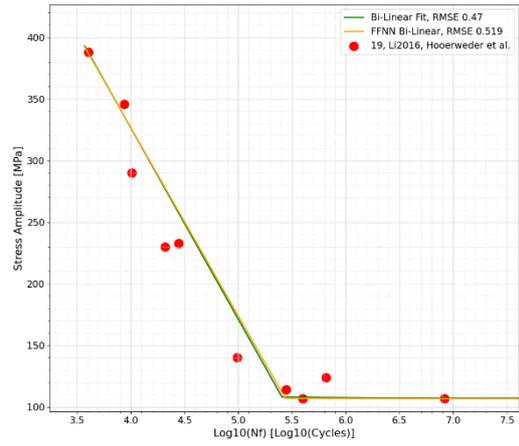
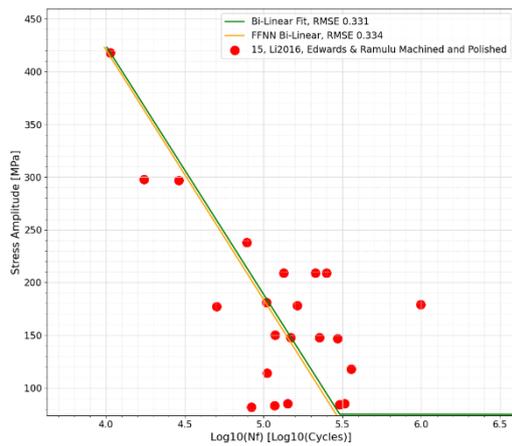
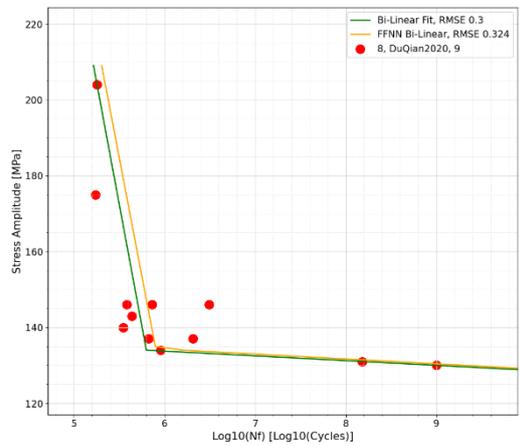
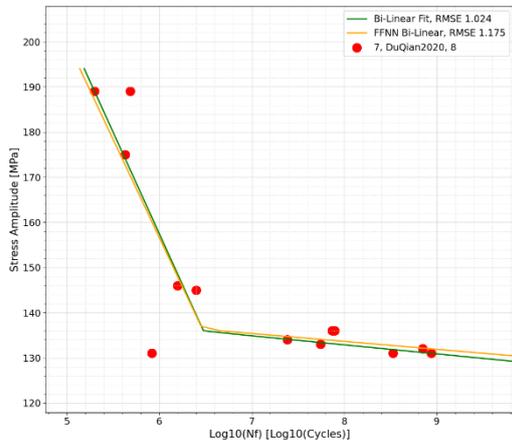
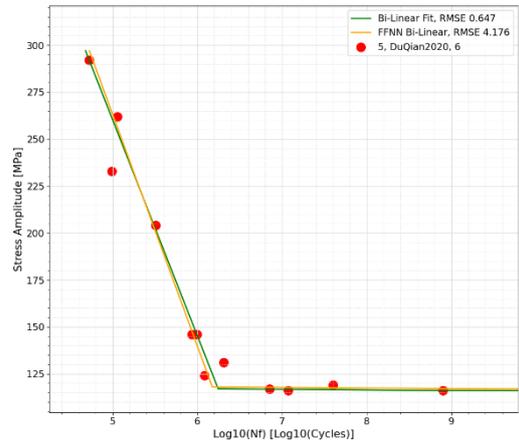
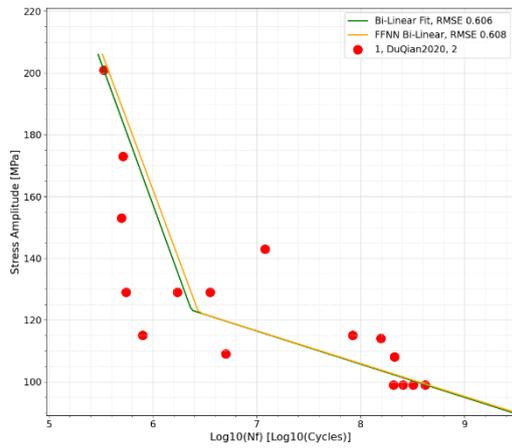


Figure 57, Usual three datasets evaluated with the 5th method: Bi-Linear F.F.N.N.

It can be clearly seen that the curves are excellent in terms of prediction, and they respect the traditional shape of fatigue problems. Other figures can be analyzed to confirm the previous statements.



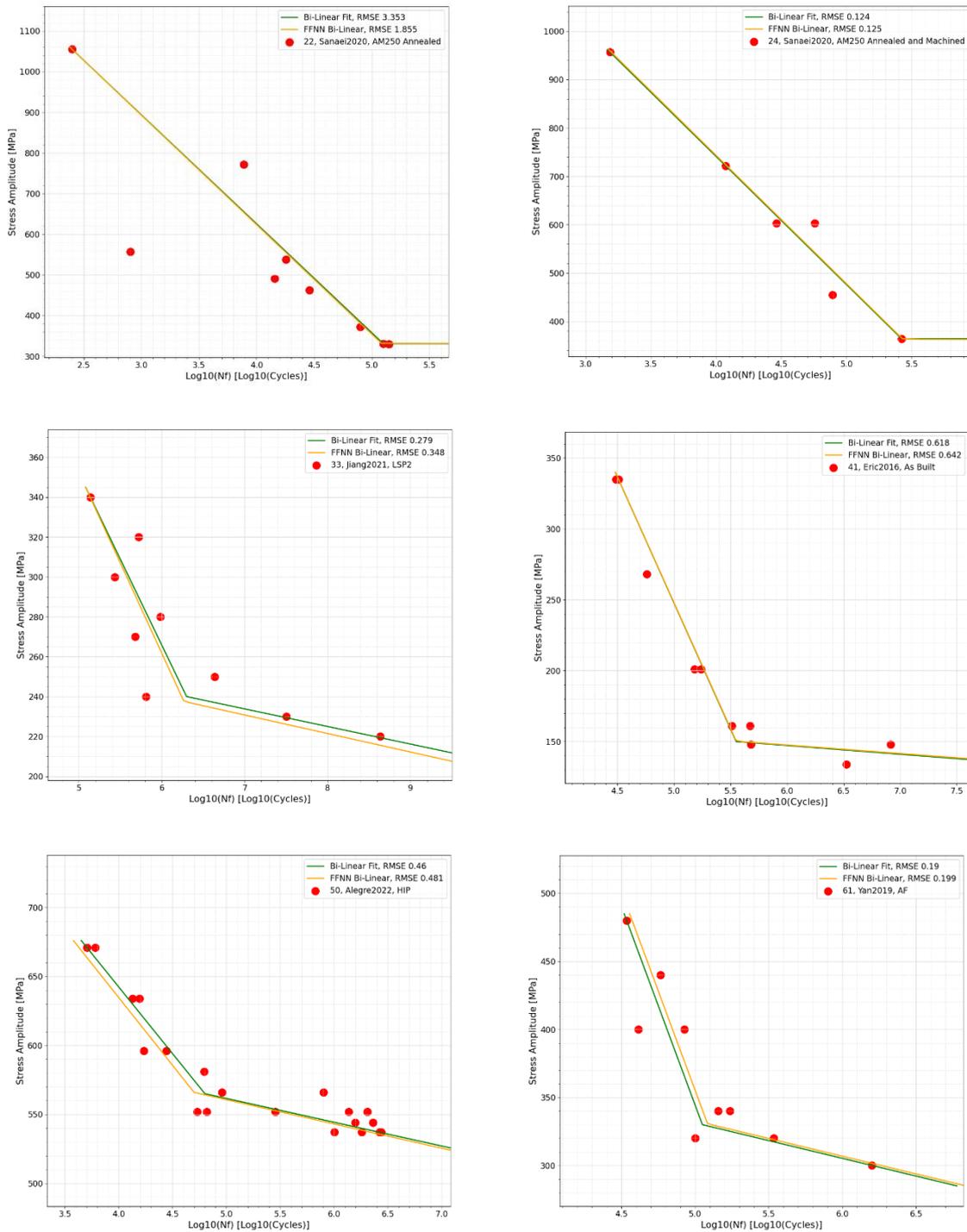


Figure 58, Datasets evaluated with the 5th type of Neural network: Bi-Linear F.F.N.N.

It can be confidently concluded that this method produces satisfactory results. It is now time to compare all the five attempts developed up to now.

## 10. Final comparison

Five types of neural networks have been developed:

1. FFNN, 3 layers ReLU (Section 5)
2. PINN, 3 layers tanh, 5 curves average,  $f'$  loss (Section 6)
3. PINN, 2 sequential layers ReLU-Linear,  $f'$  and  $f''$  loss (Section 7)
4. FFNN, 1 concatenated layer ReLU-Linear (Section 8)
5. FFNN, Bi-Linear 4 parameters (Section 9)

It is a good idea to compare them and to draw conclusions. We can first start with the comparison of the root mean square error of the five methods, that, even though it's not fully representative of the shape of the fatigue curve, it can still give us a good starting point for the final evaluation.

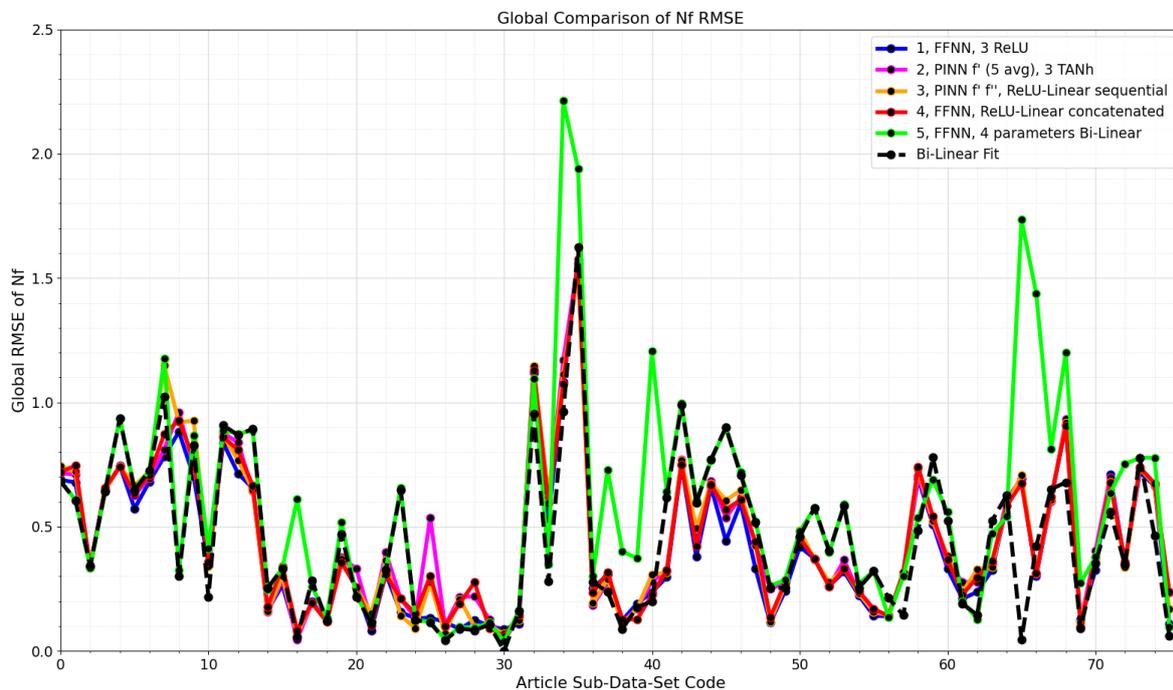


Figure 59, R.M.S.E.(Nf) for all the datasets for the 5 types of neural networks

As it can be observed, we have that the results are continuous between each other, and the discrepancies are not highly pronounced. This indicates that the methods are producing trustful results in terms of overall precision. It could be thought that the four parameters bi-linear neural network is not performing quite well as it was hoped, but, again, as it was stated in the previous sections, this numerical methodology of evaluation is not always the best option. It must be kept in mind, however, that the method developed in Section 5, has been enriched with the full database, therefore the quality of the prediction has improved significantly compared to before. By analyzing the usual three datasets we can get an idea of the situation. We can see how overall we have good predictions with all the five methods. The 3<sup>rd</sup> and 5<sup>th</sup> method grant that the concavity is always facing upward.

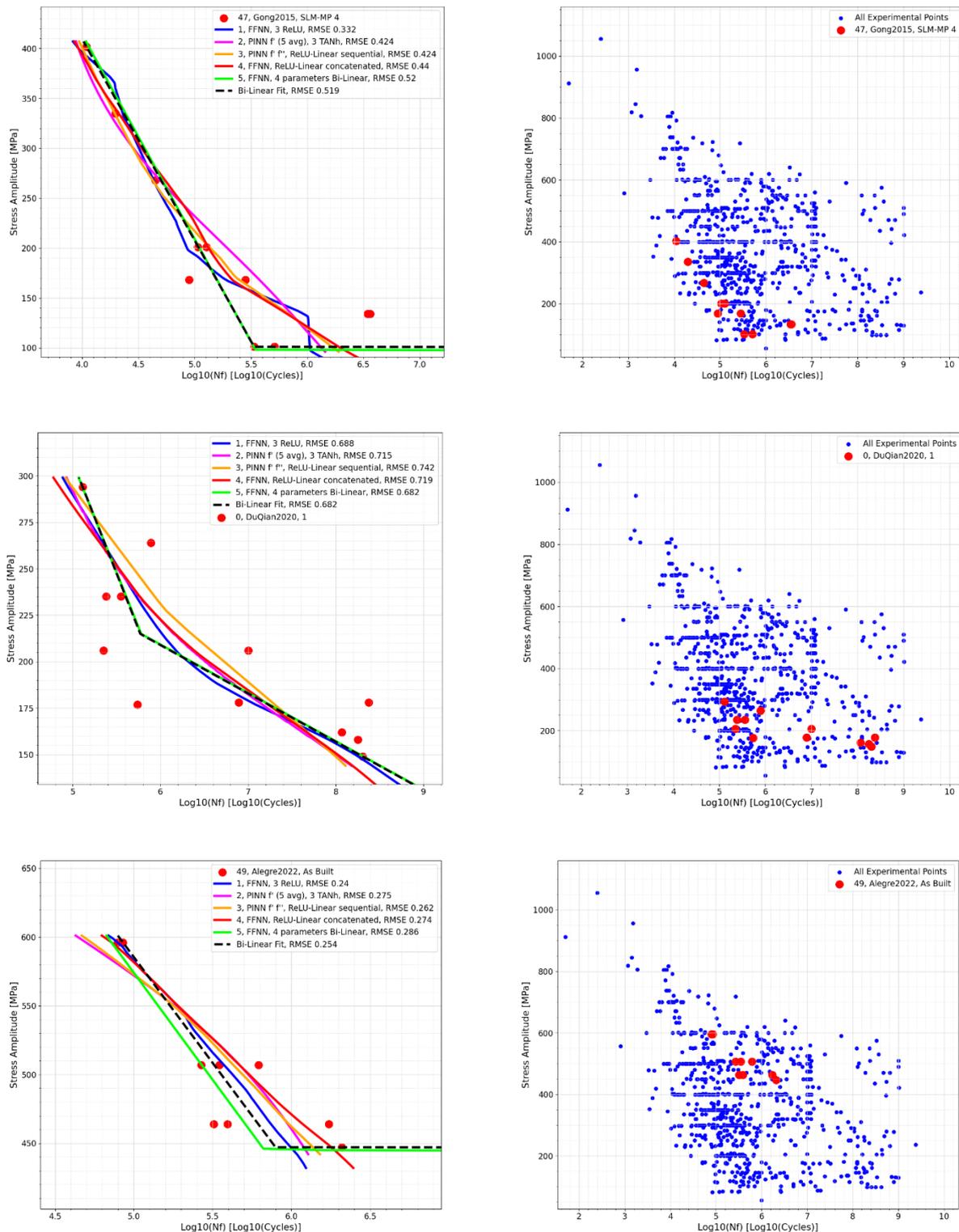
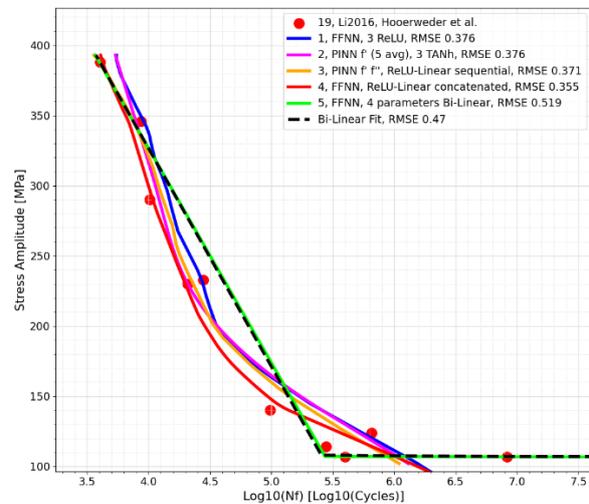
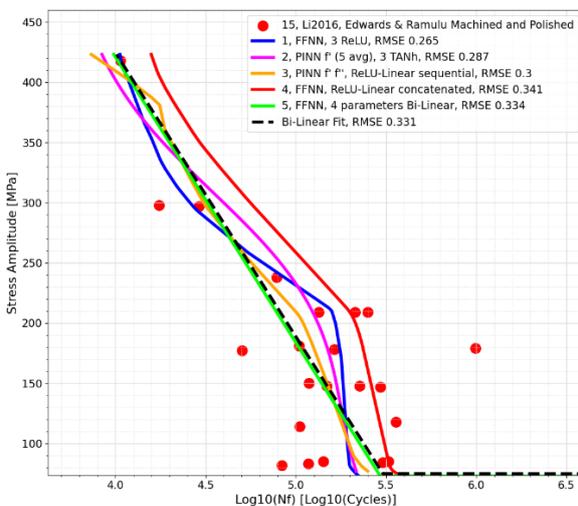
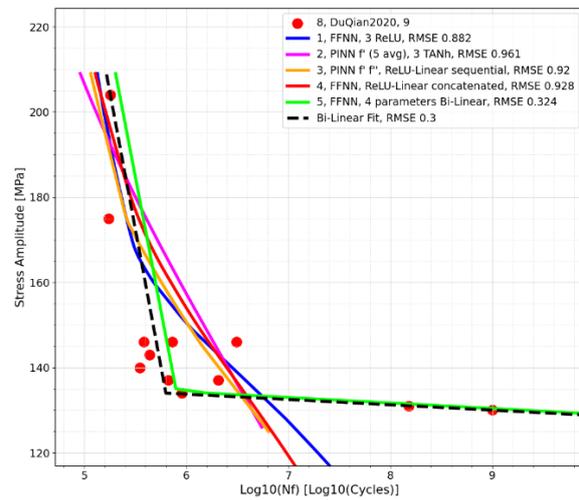
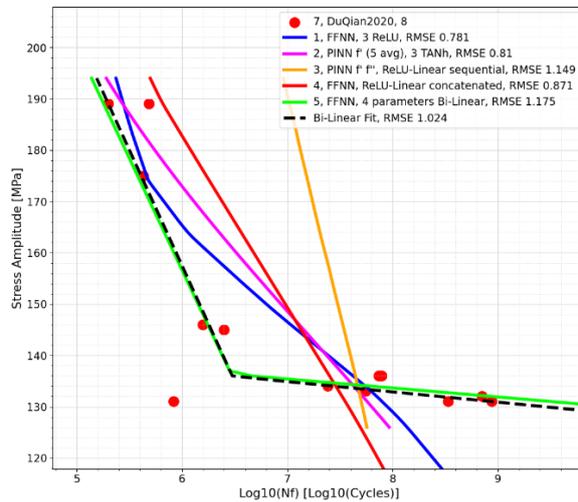
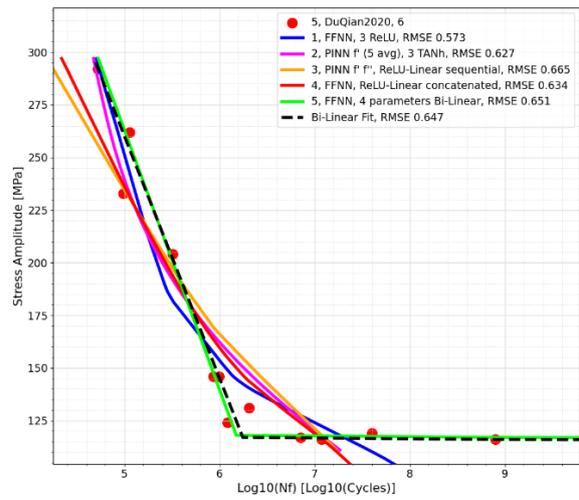
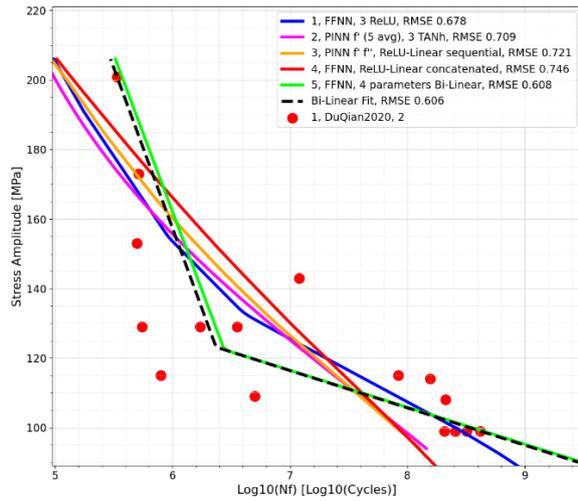


Figure 60, Comparison of the 5 methods for the 3 usual datasets

To get a deeper insight into the results, it's a good idea to visually inspect the other curves. The set of curves analyzed is the same reported in the previous sections (Figure 61).



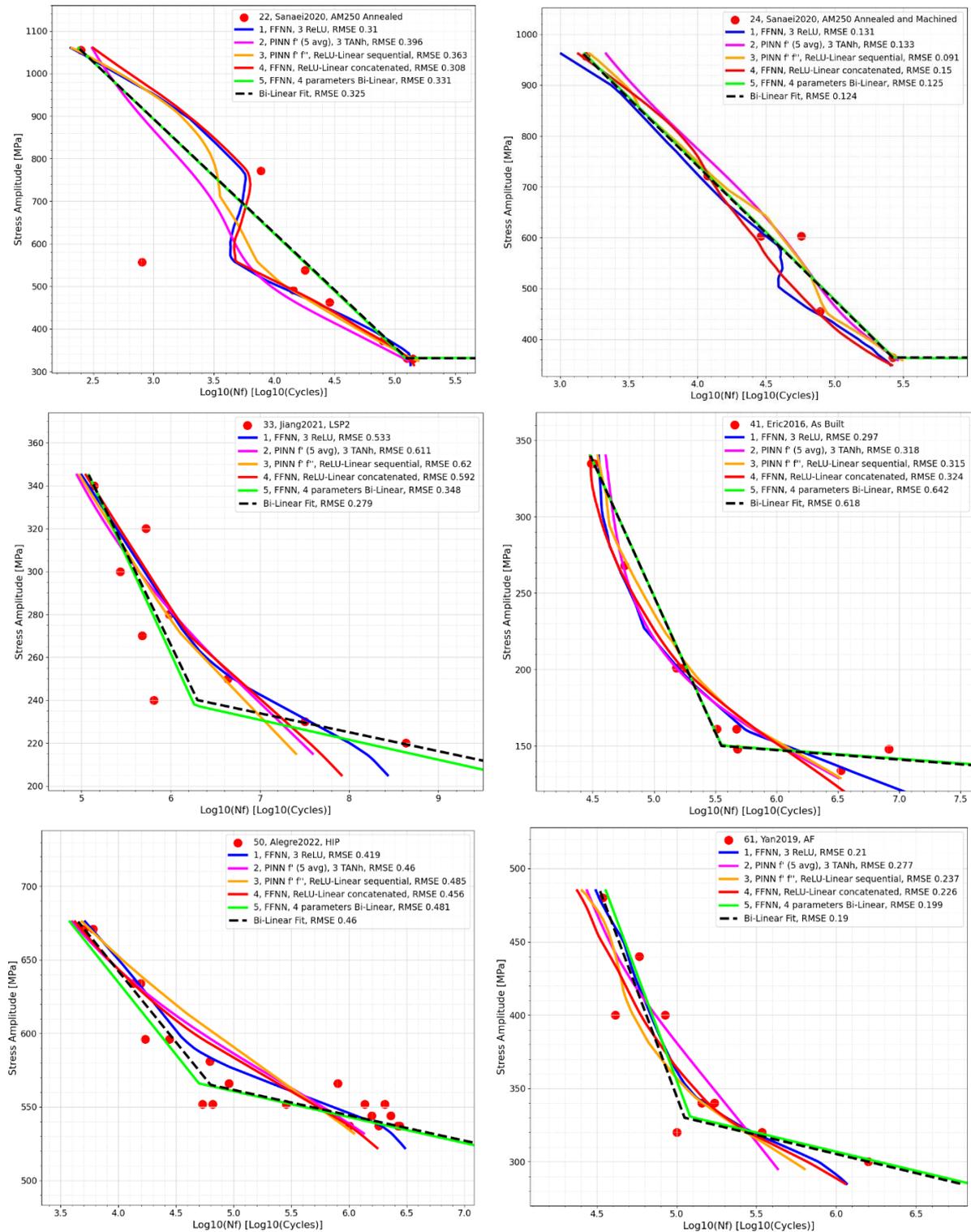


Figure 61, Datasets evaluated with all the 5 methods at the same time for comparison

As it can be seen, generally speaking, the 5<sup>th</sup> method gives curves that are better in terms of shape, concavity and strictly decreasing behavior; however, there are cases in which the overall prediction is better given by the other 4 methods. Moreover, it is important to understand what happens when we are inspecting sections of the SN plane that are out of the region of the database.

## 11. Stressing the Neural Networks

The five methodologies analyzed up to now present an overall precise behavior and the curves have an acceptable shape for almost all the subsets. Even in the extrema of the database we have a good quality of the predictions. It would be an interesting idea to see what happens if we stress the neural network by removing a decent number of points from the extrema of the database. Visually on the stress-life diagram we have this situation (*Figure 62*):

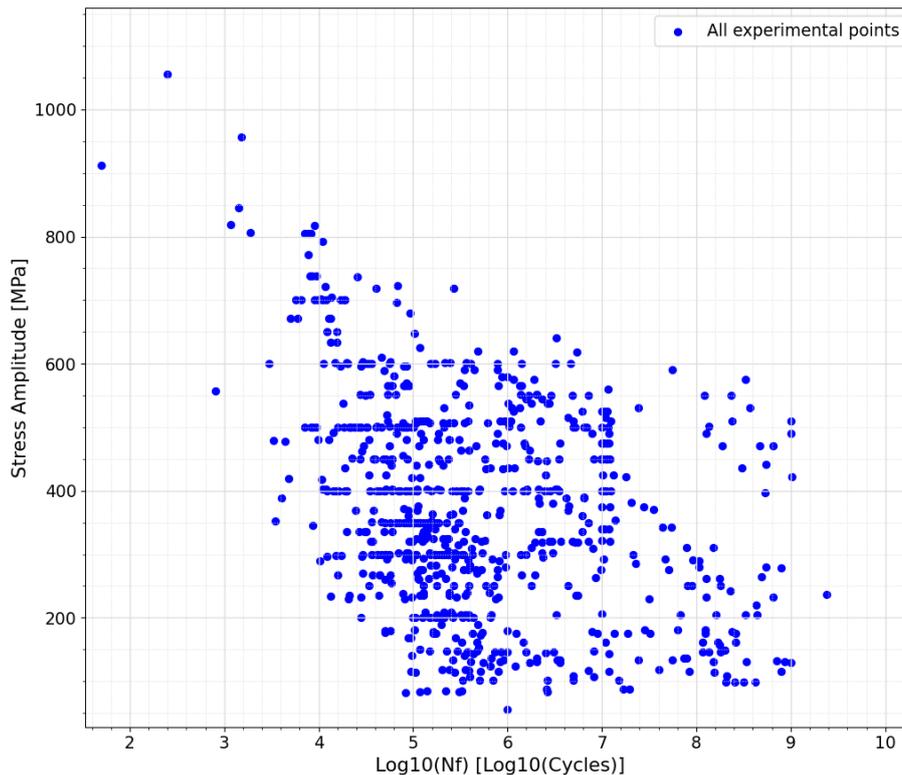


Figure 62, All the fatigue points in the S - N diagram

In total we have nearly 800 points that are spanning from stress amplitude levels of 50 MPa to 1000 MPa, while for the life we have a good variety of points that are ranging from Low Cycle Fatigue regions (L.C.F.) to Very High Cycle Fatigue life portions (V.H.C.F.). It has been decided to trim the extrema of the previous figure by removing points pertaining to the same datasets, from the training database. These datasets are belonging to extrema regions: for example, low life and low stress amplitude (*bottom left*), high life and low stress amplitude (*bottom right*), low life and high stress amplitude (*top left*) and, for how rare its occurrence is, high life and high stress amplitude regions (*top right*). 15 datasets in total have been neglected from the training, to later be evaluated in these more severe conditions. This highlights the solidity of the neural network. In fact, we can expect not so satisfactory curves evaluating the datasets that have been removed from the extrema, while it will be interesting to see what happens in the inner

portions with this smaller training database. If the points inside the inner region of the database produce approvable predictions, we can conclude that the neural network has been built on solid foundations. The training database passed from a total of 768 points to a lower amount of 620 points. *Table 13* highlights the removed datasets and their region of belonging.

<b>Code</b>	<b>Dataset Name</b>	<b>Region</b>	<b>Fatigue Properties</b>
<b>1</b>	DuQian2020, set 2	Right Bottom	High Life Low Stress
<b>7</b>	DuQian2020, set 8		
<b>14</b>	Li2016, Edwards & Ramulu As Built	Left Bottom	Low Life Low Stress
<b>15</b>	Li2016, Edwards & Ramulu Machined and Polished		
<b>19</b>	Li2016 Hooerweder et al.		
<b>22</b>	Sanaei2020, AM250 Annealed	Left Top	Low Life High Stress
<b>23</b>	Sanaei2020, AM250 AB		
<b>24</b>	Sanaei2020, AM250 Annealed and Machined		
<b>25</b>	Sanaei2020, M290 90° Annealed		
<b>26</b>	Sanaei2020, M290 45° Annealed		
<b>27</b>	Sanaei2020, M290 90° AB		
<b>28</b>	Sanaei2020, M290 45° AB		
<b>12</b>	Gunther2017, SLM-2	Right Top	High Life High Stress
<b>34</b>	Sun2021, Ultrasonic		
<b>35</b>	Sun2021, Rotating Bending		

*Table 13, Strategy used to stress the Neural Networks*

To have a visual representation of the situation after the removal of the points we have *Figure 63*, that shows all the points of the original database, with the removed points highlighted in red.

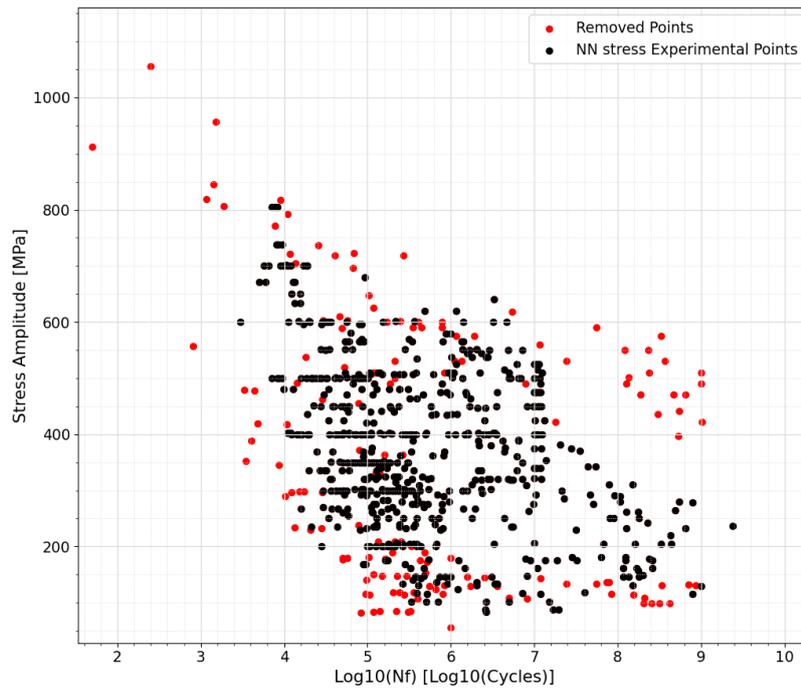


Figure 63, Training database with the points to be removed highlighted in red

Therefore, the final database is significantly narrowed by this removal and its final look is the one shown in the figure below (Figure 64).

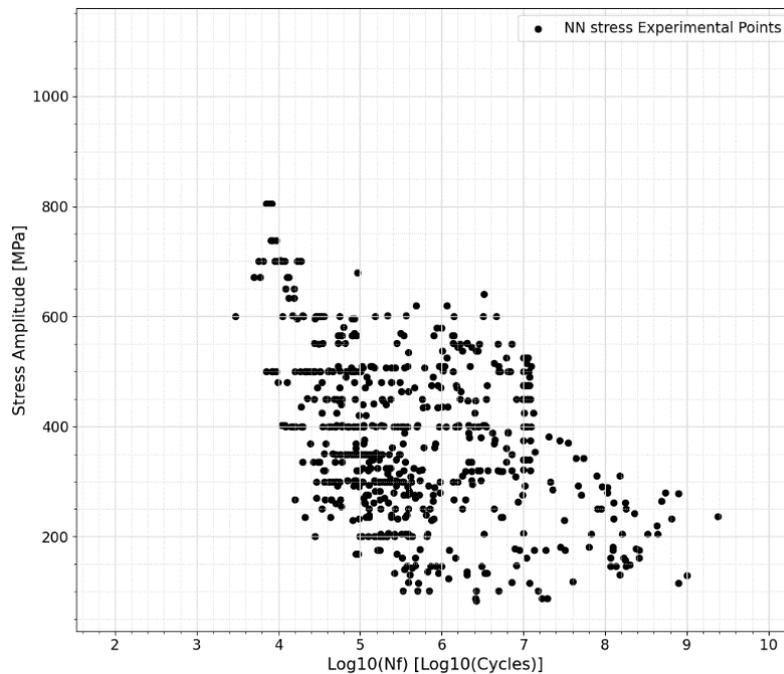


Figure 64, Trimmed training database used to stress the neural network

### 11.1 Inspecting the Removed portions

It's important to study the parts of the SN plot, from which we have removed the points, to see how strong their predictions are. Reporting all the 15 datasets listed in *Table 13*, with all the methods compared, we can get an idea on how the 5 neural network types are having difficulties in getting decent predictions.

#### Right Bottom

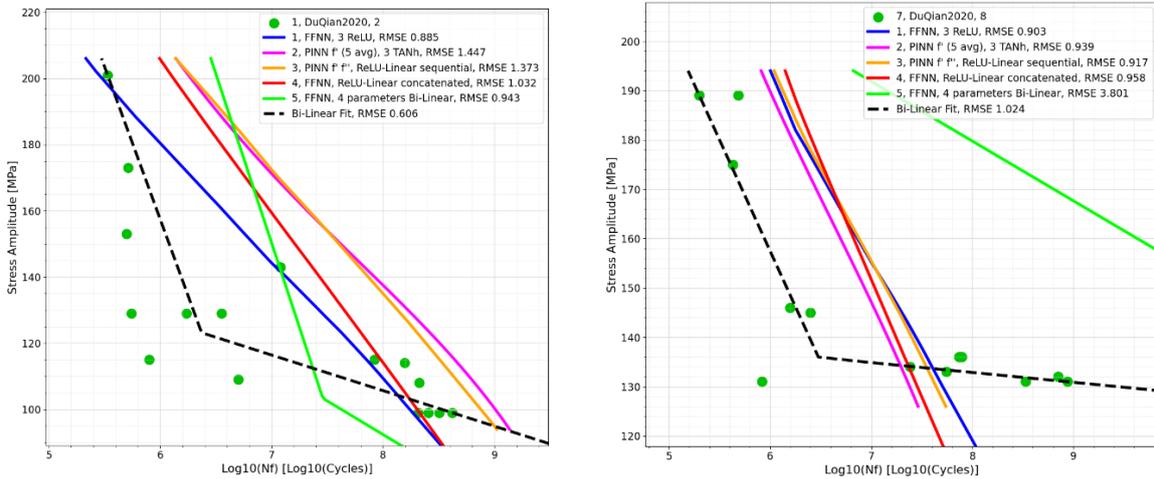
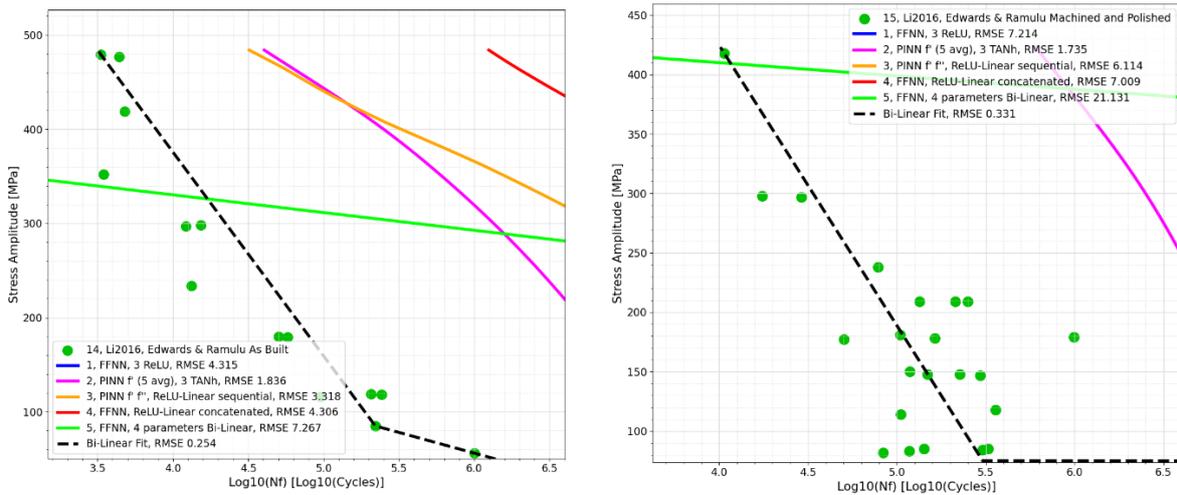


Figure 65, Right bottom region stressed network analysis

For the right bottom section, we are having that all the 5 types of neural networks are getting right the portion of the experimental points. However, we can see that the overall precision is not extremely good. It has to be noted that the 5th method (bilinear prediction) is performing worse with respect to the others.

#### Left Bottom



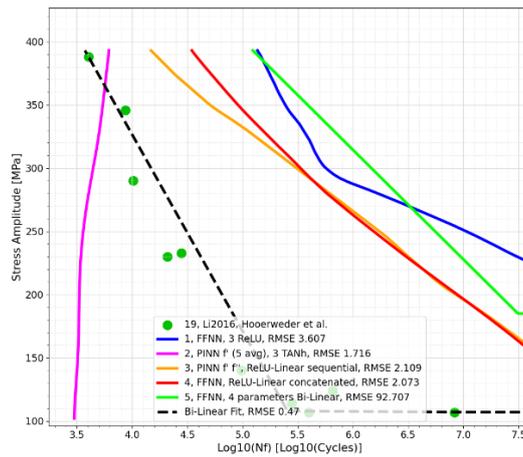
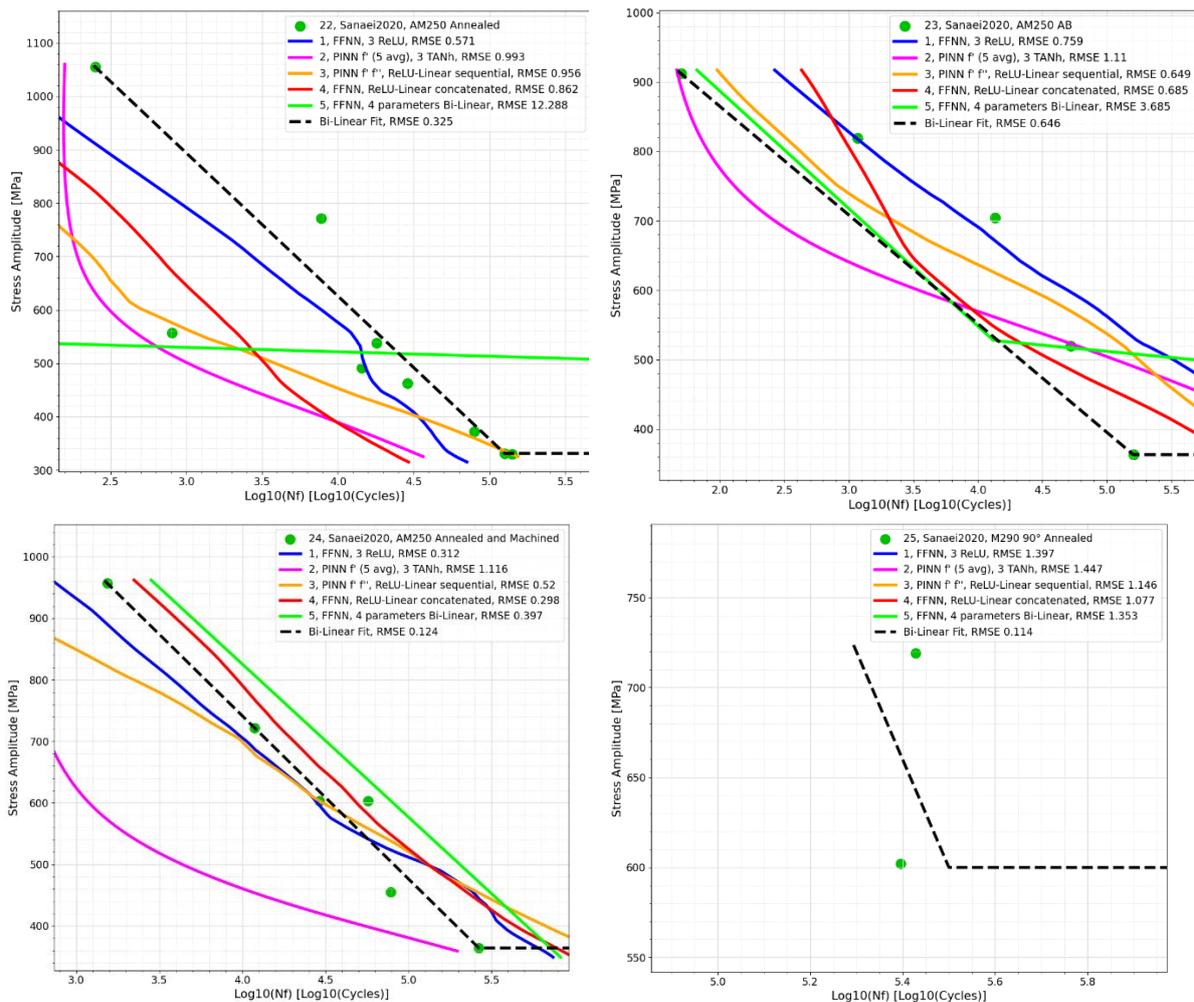


Figure 66, Left bottom region stressed network analysis

For the left bottom region, we have terrible curves. This happens because in those regions the database is poorly populated, therefore the neural network is not able to grasp any significant path to get a correct prediction.

### Left Top



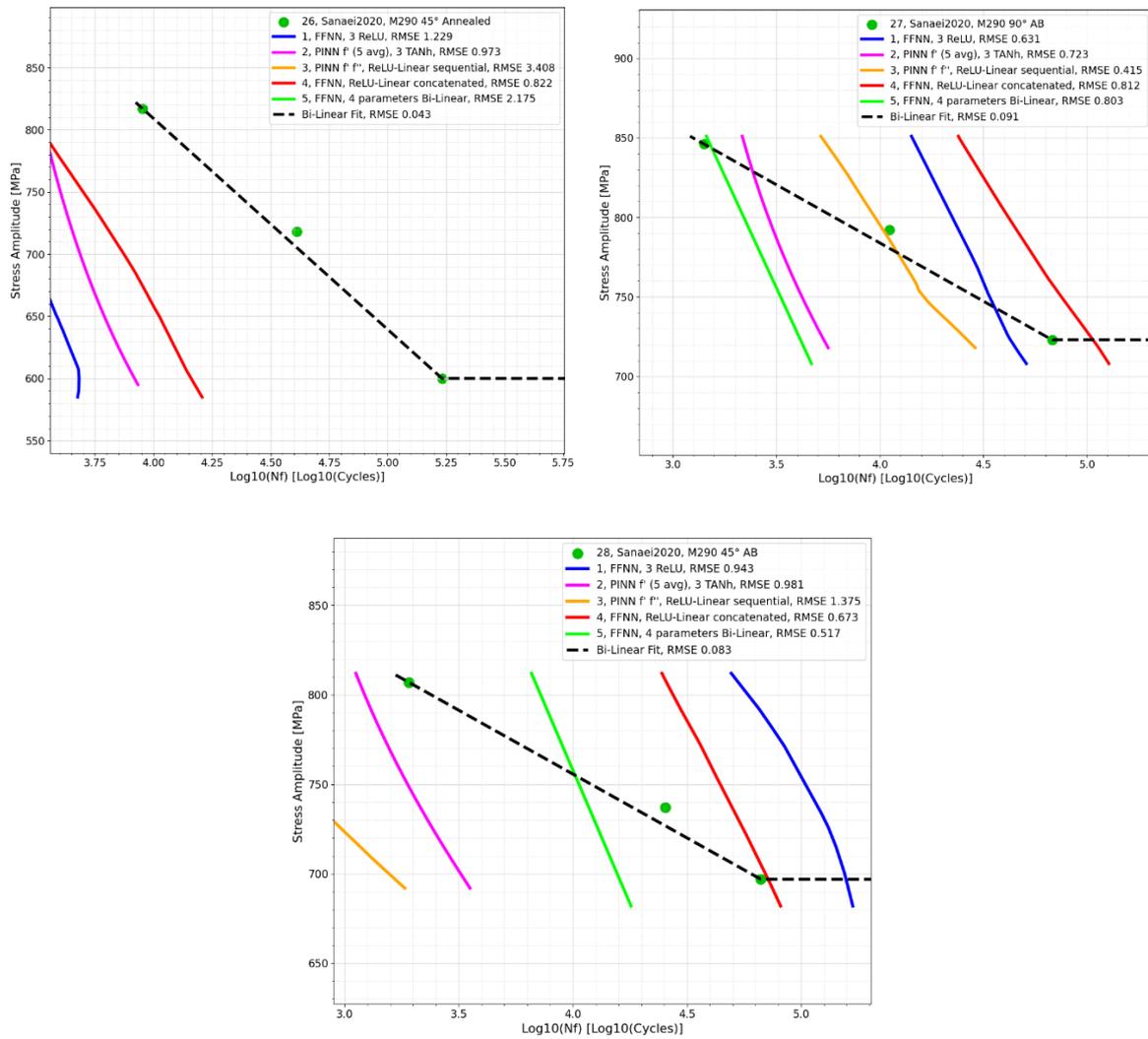


Figure 67, Left top region stressed network analysis

If we switch to the left top part, we have the most controversial outcome of the 4 areas. In fact, for three of the figures we have almost precise predictions, while for another couple of them we are getting right just the portion of the experimental points, but we are missing the right slope, or we are having a significant offset of the curve with respect to the points. For another couple of curves, the prediction is completely out of range. Again, it's worth noting that the bilinear method is having more difficulties with respect to the other 4 techniques.

Right Top

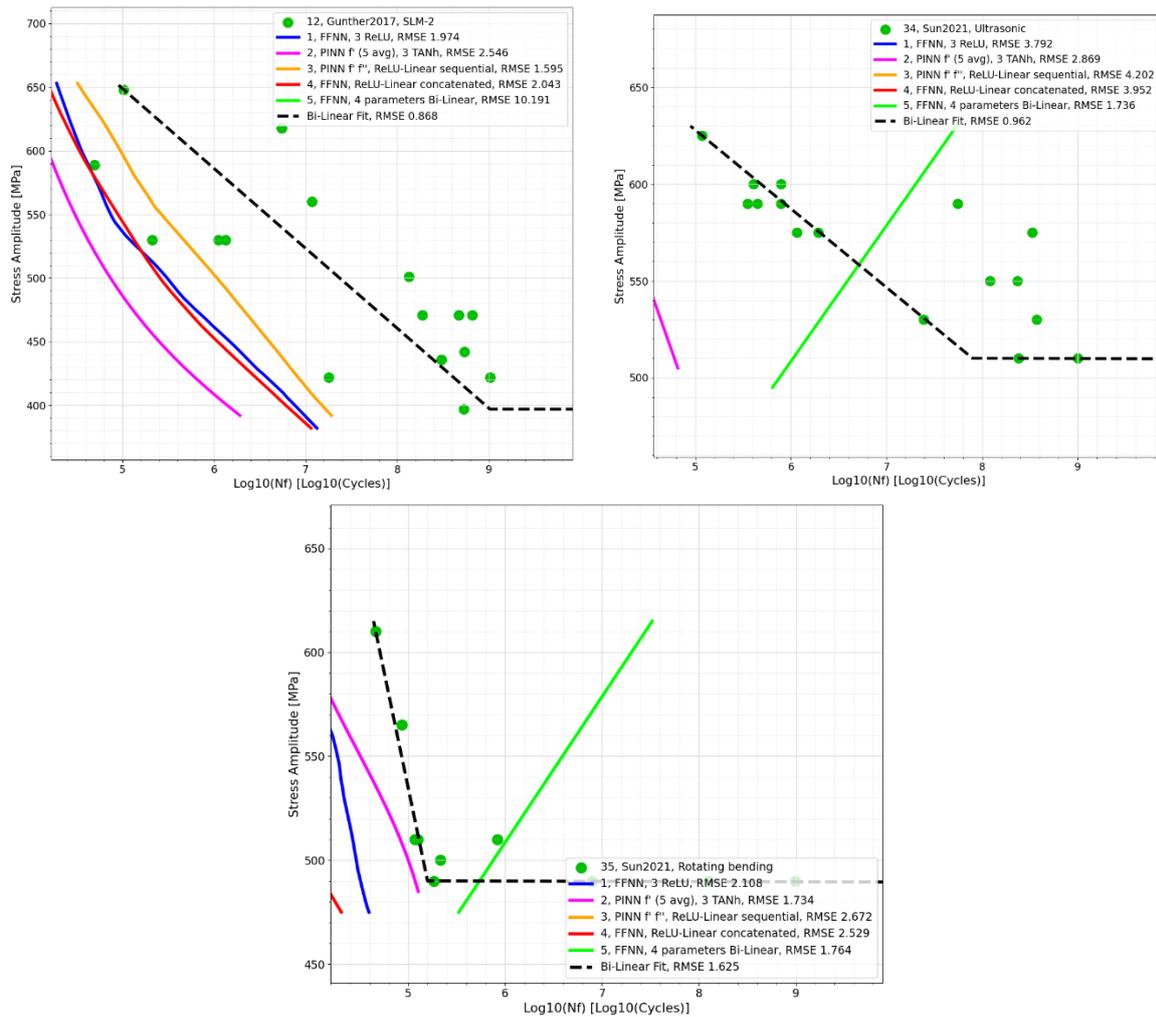


Figure 68, Right top region stressed network analysis

Again, as it happened in the left the bottom part, here we are having disastrous outcomes. The reason has again to be searched in the physics of fatigue problems. The majority of the points are found in a sort of linear region that is decreasing from the top left region to the bottom right region. Therefore, it's more likely to find points in those parts, while the occurrence of failures in the bottom left or top right region is seldom. This explains why in those two regions the predictions are bad: because the database in those areas is not richly populated.

## 11.2 Analyzing the Inner Portions

Since reporting the missing 61 curves is not practical, first the three usual preliminary datasets “*Gong2015 SLM-MP 4*”, “*DuQian2020, set 1*” and “*Alegre 2022, As Built*” will be shown and then the remaining other curves, that have been shown multiple times in the previous sections, will be inspected.

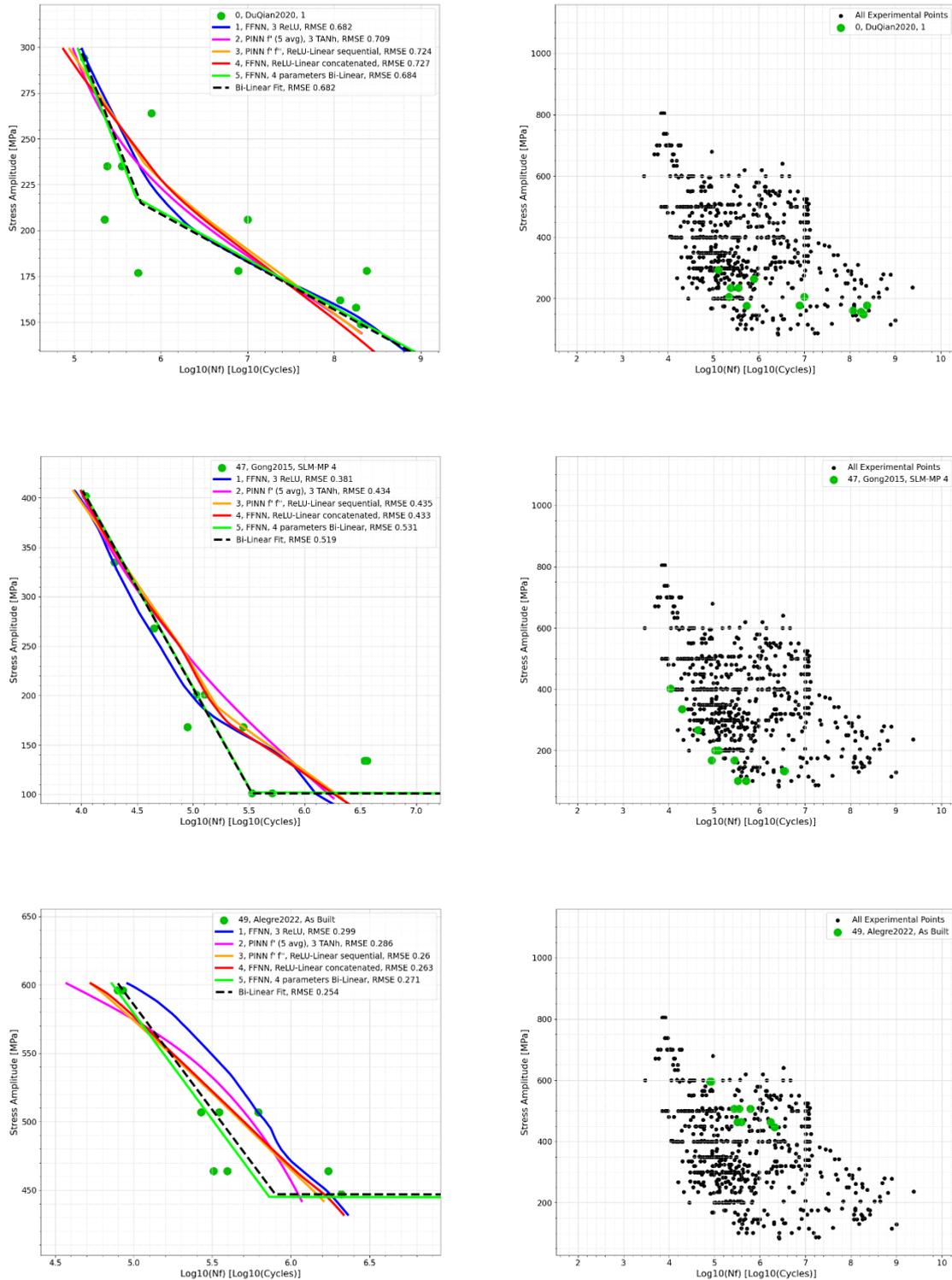
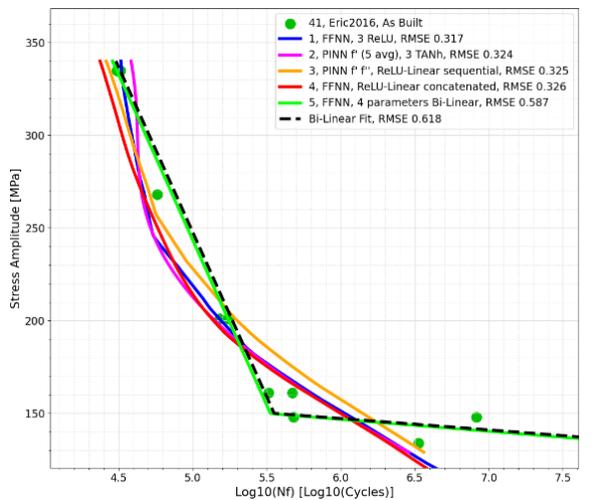
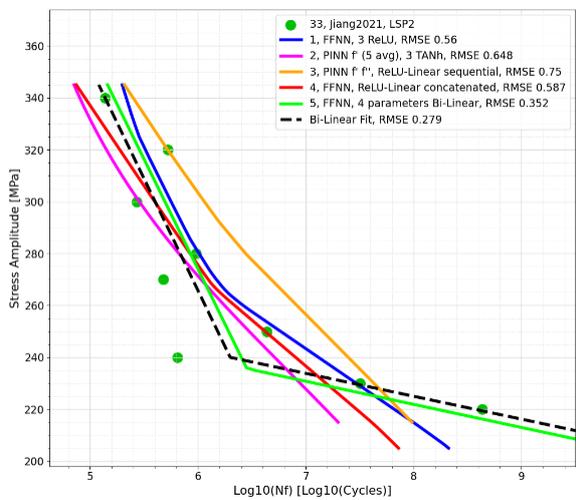
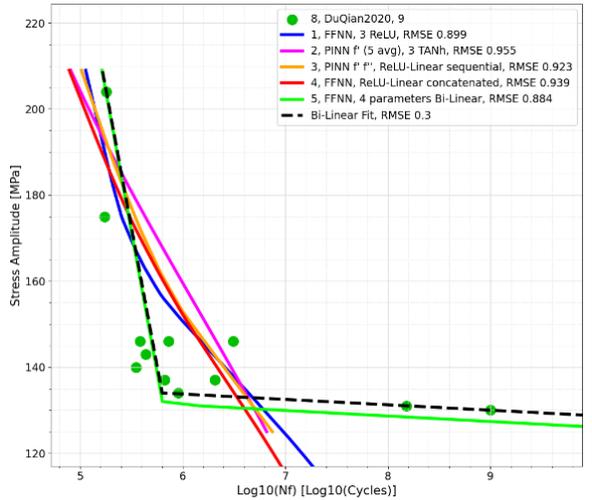
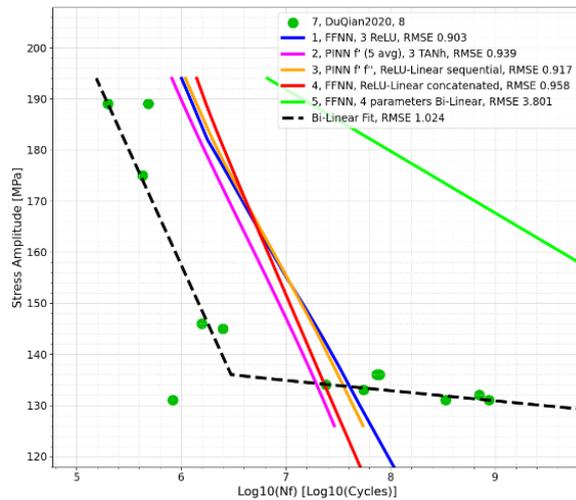
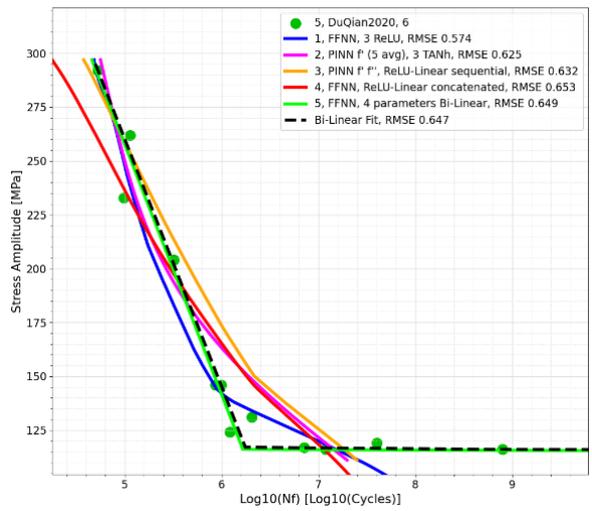
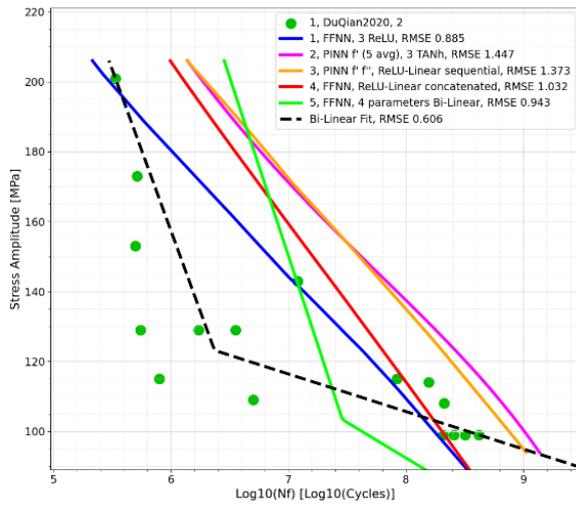


Figure 69, Usual three datasets analysed in the stressed condition

At first glance it can be confidently stated that the predictions of this neural network stress test are very similar to the ones obtained in *Section 10*. To confirm this statement, it's an excellent idea to look at the other curves.



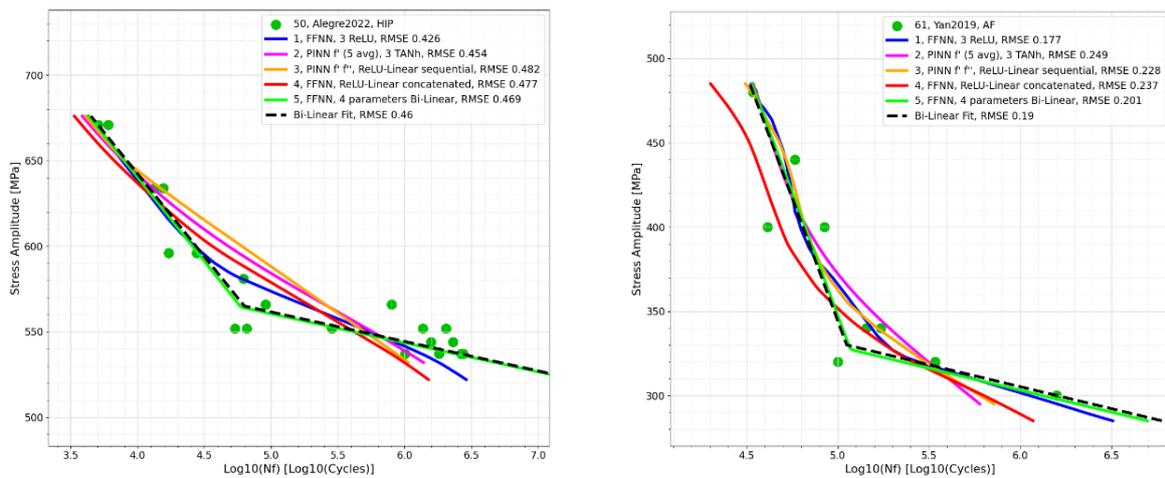


Figure 70, Datasets in the inner region evaluated in the stressed conditions

Indeed, the previous statement is true: as it was observed for the three usual datasets, the quality of the prediction is almost identical to the one of *Section 10*. The fact that we are getting good predictions in the inner region, is a sign that the method that has been implemented in the neural network is coherent and solid.

In conclusion, the curves of the inner portions are not influenced by the outer regions, while, if we are inspecting sections of the SN plane that are out of the region of the database, the 4-parameters bi-linear method has difficulties in outputting a decent curve, while the other 4 methods, can still guess a portion of the region.

## 12. External Validation

Since in the previous sections, the fatigue neural network has been demonstrated to be working nicely, delivering correct predictions, it is definitely a good idea to make sure that the network can deliver viable predictions in the case it is used as a “program” to output the fatigue curves for specimens produced with a determined set of process parameter. Since the specimens are not yet produced and tested, and since the aim of this work is to avoid the production and the testing of specimens, we cannot insert other experimental points inside the training database. We could potentially ask the network to output curves for a set of process parameters, to see in which area of the SN diagram the curve belongs, but we wouldn't have any way to prove if the prediction is correct. To this end, to emulate this situation, and to prove that the network is capable of delivering correct predictions, an external article has been searched to create a validation dataset. After extensive research, the article “*Xu2020*” [23] has been found. It has three sets with 20 fatigue points circa, belonging to specimens produced with the same sets of process parameters, but with respectively a varying building orientation (90°, 45° and 0°). *Table 14* resumes the values of the process parameters and thermal and surface treatments, besides the coordinates of each point (*Table 15*).

Features	Set 1	Set 2	Set 3
Building Orientation [°]	90	45	0
Laser Power [W]	450	450	450
Hatch distance [mm]	0.15	0.15	0.15
Scan speed [mm/s]	1200	1200	1200
Layer Thickness [µm]	50	50	50
Annealed?	Yes	Yes	Yes
Annealing Temperature [°C]	850	850	850
Hot Isostatic Pressing?	No	No	No
Machined?	Yes	Yes	Yes
Sand Blasted?	No	No	No
E.D.M.?	No	No	No
Laser Shot Peening?	No	No	No
Shot Peening?	No	No	No
S.M.A.T.?	No	No	No
Polished?	Yes	Yes	Yes

*Table 14, Features for the external validation article*

	Set 1		Set 2		Set 3	
	$\sigma_a$ [MPa]	$Log_{10}(N_f)$	$\sigma_a$ [MPa]	$Log_{10}(N_f)$	$\sigma_a$ [MPa]	$Log_{10}(N_f)$
1	843	4.42656	750	4.173186	802	4.397801
2	758	4.565942	674	4.537618	850	4.649637
3	590	4.765162	675	4.710227	590	4.860895
4	547	4.86163	590	4.943445	548	4.906922
5	589	4.866618	590	5.021603	758	4.913687
6	547	4.891666	506	5.124765	674	4.923358
7	674	4.98209	590	5.153205	506	5.165304
8	463	5.053731	548	5.15791	548	5.166963
9	505	5.055875	440	5.215373	590	5.446522
10	506	5.10394	548	5.233453	674	5.450172
11	506	5.138208	506	5.242094	589	5.502004
12	463	5.160589	505	5.266537	464	5.598156
13	590	5.201315	421	5.444872	506	5.60608
14	564	5.238071	421	5.520732	548	5.72125
15	463	5.725593	440	5.661197	464	6.885022
16	548	7.015779	404	5.843432	674	7.125188
17	565	7.464668	404	5.903535	590	7.202379
18	421	7.500826	337	7.991536	548	7.31029
19	380	7.990516	378	7.994625	464	7.629522
20	338	7.993304	397	7.996643	506	7.660619
21					589	7.662956
22					506	7.863424

Table 15, Experimental points for the 3 datasets of the article Xu2020

Herebelow are reported the three datasets besides the global map, to identify their region of belonging. A good number of points, are located in poorly populated regions. This means that if we get decent predictions, the methods developed until now are consistent; because it's more difficult to get good results in scarce regions.

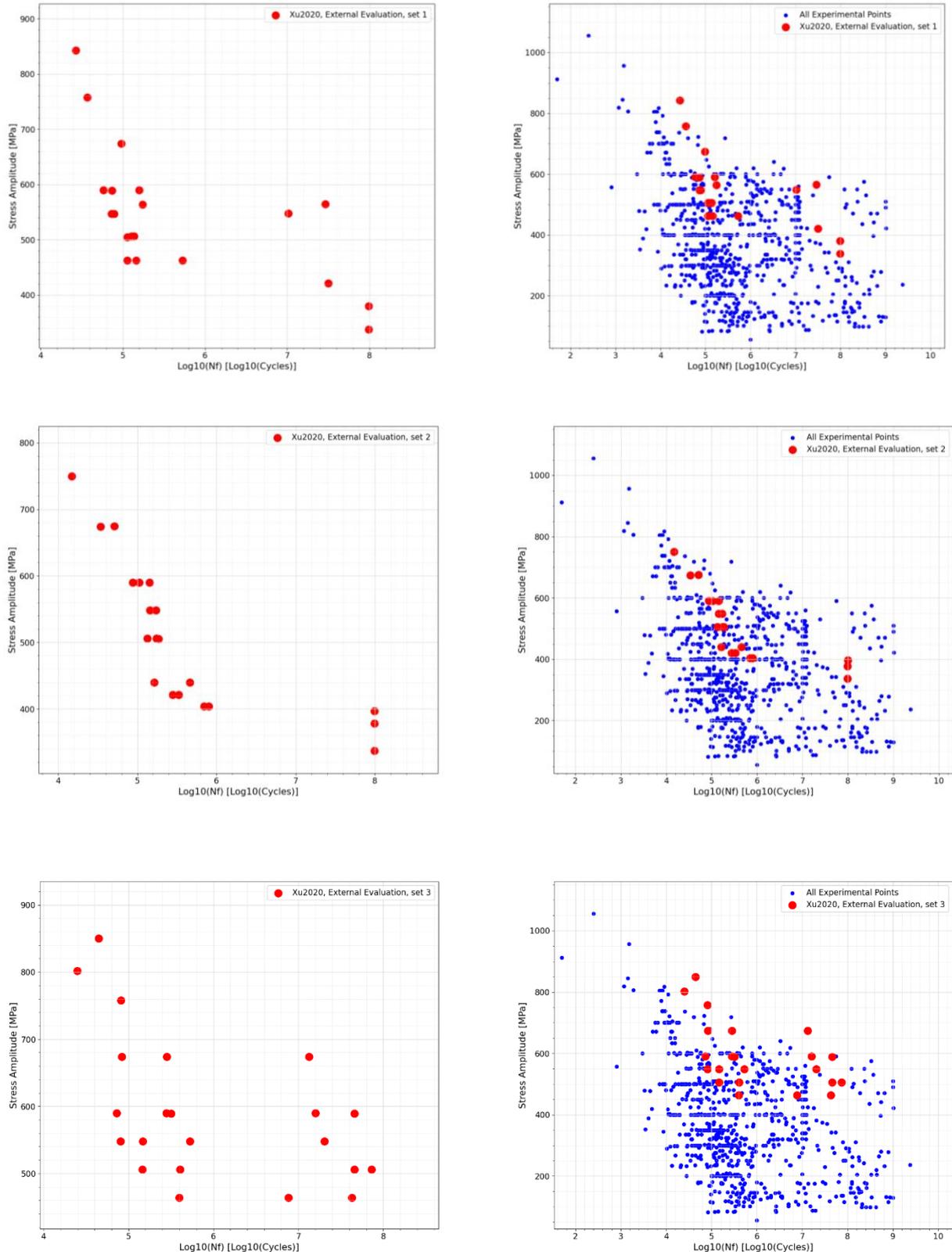
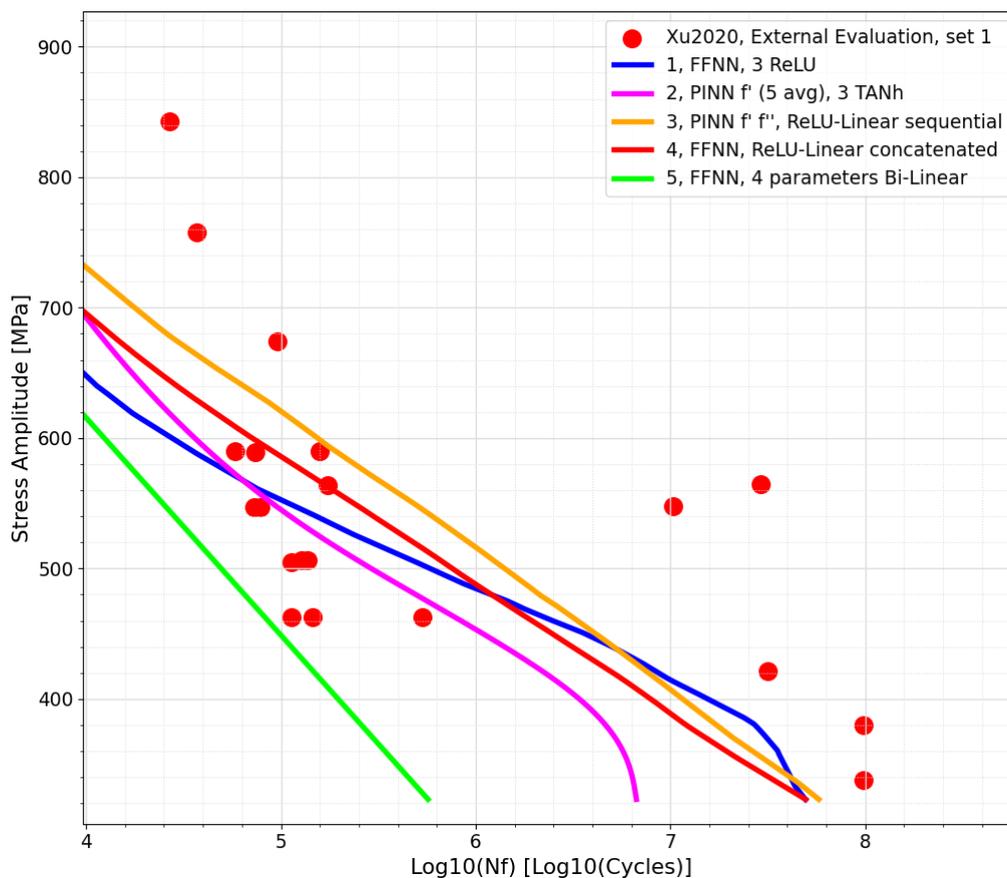


Figure 71, The 3 datasets of the article Xu2020

The curves have been evaluated with the usual five methods. The first set produces decent results for the first 4 methods, as the middle and low sections are covered. The upper part has a low-quality prediction. This is due to the fact that the 3 points that have a high stress amplitude belong to a region that has not been trained yet. Therefore, such behavior is expected. The 5th method, although is the most conservative one, is not capable of covering the points.

The second set is the one that has the best predictions of all. The reason has to be searched again in the fact that those points pertain to regions that have been already explored during the training. The best shapes are achieved by the 4th and 5th method. The other three are able to land the curve on the points, but the shape is not so convincing.

For the third set, we are having that the bilinear method delivers the best prediction, followed immediately by the second method. The first method is also producing a nicely shaped curve, though it's too much on the far left. The 4th one provides a suitable shape, but doesn't give a correct enough prediction, as the points are not guessed.



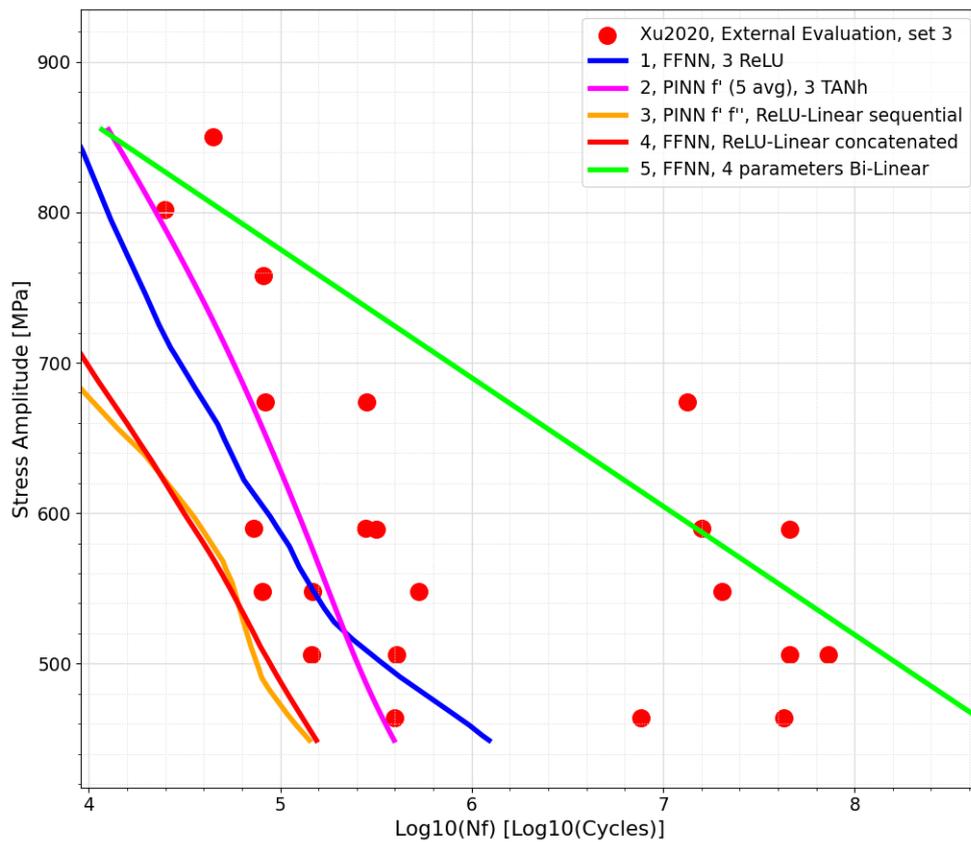
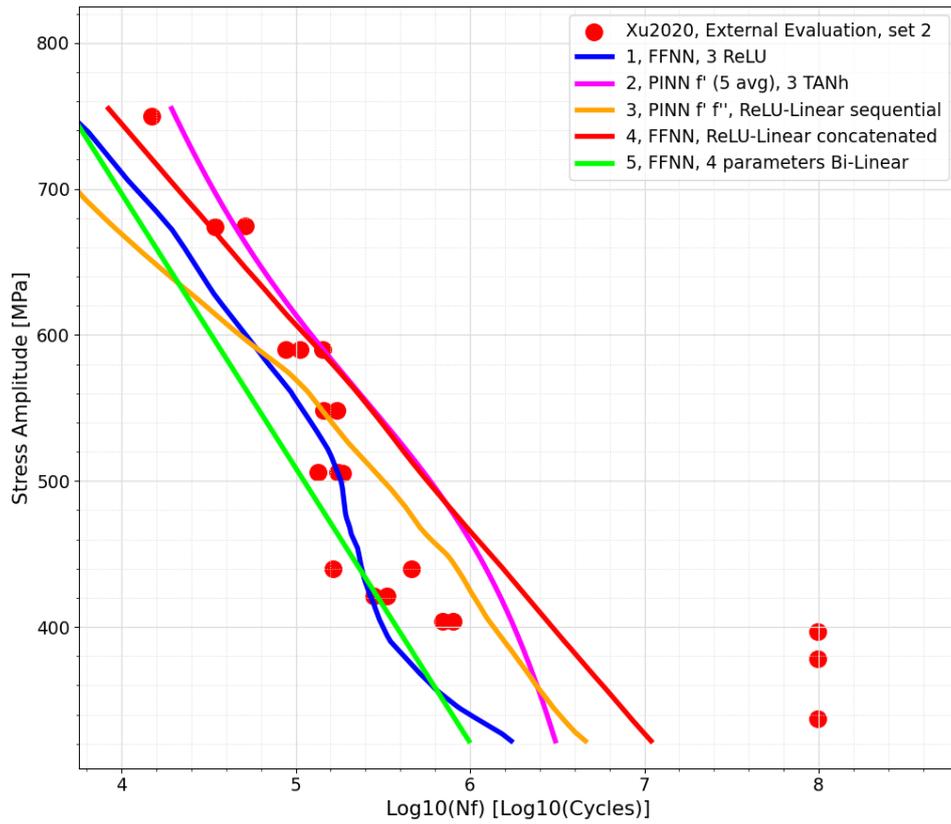


Figure 72, The 3 datasets of the article Xu2020 evaluated

Even if the predictions are not perfect, we can state that the previous figures are a sign of consistency when it comes to evaluating the goodness of the neural network with external data. Moreover, the fact that those points belong to poorly populated regions is a further sign of the solidity of the network, as it can perform well enough even in these scarcely populated conditions. When evaluating a fatigue curve with a set of process parameters with these methods, we can be confident enough that we can produce a viable prediction.

### 13. Process Parameter Variation Evaluation

Having asserted that the neural network is trustful, we can now proceed with one of the last evaluations. We can detect what is the overall trend of the neural network, considering that each process parameter varies while keeping the others constant. This gives us an additional reason to trust what has been developed up to now. In fact, we can believe in the correctness of the method if we see that, by varying one of the process parameters, the fatigue curve changes accordingly and reasonably with respect to the physics of the selective laser melting problem. To proceed with this evaluation, a feedforward neural network with only one linear layer with 1000 neurons, has been chosen. The following strategy allows us to have a simple, yet effective, representation of the results, as the resulting shape is a simple line that will not intertwine with the other lines that are going to be plotted around. The line will be evaluated in the entire stress-life diagram. The X axis ranges from  $10^1$  to  $10^{10}$  cycles, while the Y axis spans stress amplitudes from 0 MPa to 1000 MPa. With this it's possible to have a complete global vision. It has been chosen to vary the ***building orientation***, ***hatch distance***, ***layer thickness*** and the ***speed/power*** cross parameter. Therefore, we are having four separate evaluations and while we are varying one of the process parameters, the other ones are kept constant to a precise value that is the same for each of the four evaluations. To be even more precise by laying down curves that are located in the mid region of the fatigue diagram, it has been chosen to have all the process parameters and the booleans of the thermal treatments and surface treatments set to the most recurrent or mean value considering the whole database, if the feature allows to do so. Since, again, the neural network is outputting as predicted label, the fatigue life  $N_f$ , we are using a vector of stress amplitude values that is varying from 50 MPa to 1000 MPa with steps of 25 MPa. For the ***building orientation*** parameter, we have that it can just have three values:  $0^\circ$ ,  $45^\circ$  or  $90^\circ$ ; therefore, it's useless to perform a mean value, as  $80^\circ$  is never used to manufacture S.L.M. parts. In this case the most frequent value of  $90^\circ$  has been chosen. For the ***speed/power*** cross parameter, since it derives from the combination of two numbers it can have continuous values from its minimum to its maximum, hence it's a viable choice performing its mean, that is  $4.8 \text{ mm/s} \cdot \text{W}$ . The ***hatch distance*** ranges from more or less 0.05 mm to 0.25 mm. Again, as before, we can have continuous values from the minimum to the maximum, so the mean value of 0.12 mm has been adopted. The same discussion holds for the ***layer thickness***: the minimum is 30  $\mu\text{m}$  while the maximum is 60  $\mu\text{m}$ ; the mean value performed on the whole database is 40  $\mu\text{m}$ . Concerning the boolean values, since it doesn't make sense to perform a mean value between 0 and 1, also considering its logical nature, the most recurrent value has been considered. This results in a zero for all the

booleans except the *annealing*, *machining* and *polishing* treatments, that are the most frequent ones. Since the annealing process has been included for the aforementioned reason, the annealing temperature must not be set to 20 °C. So, the mean value of all the annealing temperatures, excluding 20 °C, has been performed resulting in 735 °C. *Table 16* clarifies what has been just explained.

	D	E	F	G	I	J	K	L	M	N	O	P	Q	R	S
1	90	4.8	0.12	40	1	735	0	1	0	0	0	0	0	1	50
2	90	4.8	0.12	40	1	735	0	1	0	0	0	0	0	1	75
...															
i	90	4.8	0.12	40	1	735	0	1	0	0	0	0	0	1	...
...															
N	90	4.8	0.12	40	1	735	0	1	0	0	0	0	0	1	1000

D	Build Orientation [°]	L	Machined Boolean
E	Scan Speed / Power [mm/s·W]	M	Sand Blasted Boolean
F	Hatch Distance [mm]	N	E.D.M. Boolean
G	Layer Thickness [µm]	O	L.S.P. Boolean
I	Annealed Boolean	P	S.P. Boolean
J	Annealing Temperature [°C]	Q	S.M.A.T. Boolean
K	H.I.P. Boolean	R	Surface Polishing Boolean
		S	Stress Amplitude [MPa]

Table 16, Process Parameter variation evaluation database

Once the evaluation database is built, we can repeat it as many times as it is needed to cover each process parameter variation. If we choose, for example, to vary the *building orientation*, the database is repeated three times, but for the first set, 0° will be the value under the build orientation column, for the second we will have 45° and for the third 90°, while keeping all the other parameters set to the decided values, displayed in the previous table. This process is repeated several times up until all the process parameters have been covered, with its chosen variation range.

### 13.1 Building Orientation

Having anticipated that the building orientation varies from values equal to 0°, 45° and 90°, it's trivial to choose to let this parameter vary with these three values. If we run the training and we evaluate it with these three values and we combine the obtained figures on the whole database, we obtain *Figure 73*. As it can be easily seen, the 90° configuration outputs the best curve, followed immediately after by 45° and 0°. This is a sign that specimens built vertically are performing

better with respect to horizontally built ones. As it was anticipated before we can see that the curves belong to the mid region, indicating that having chosen the mean sets of process parameters leads to the plot being located in the center.

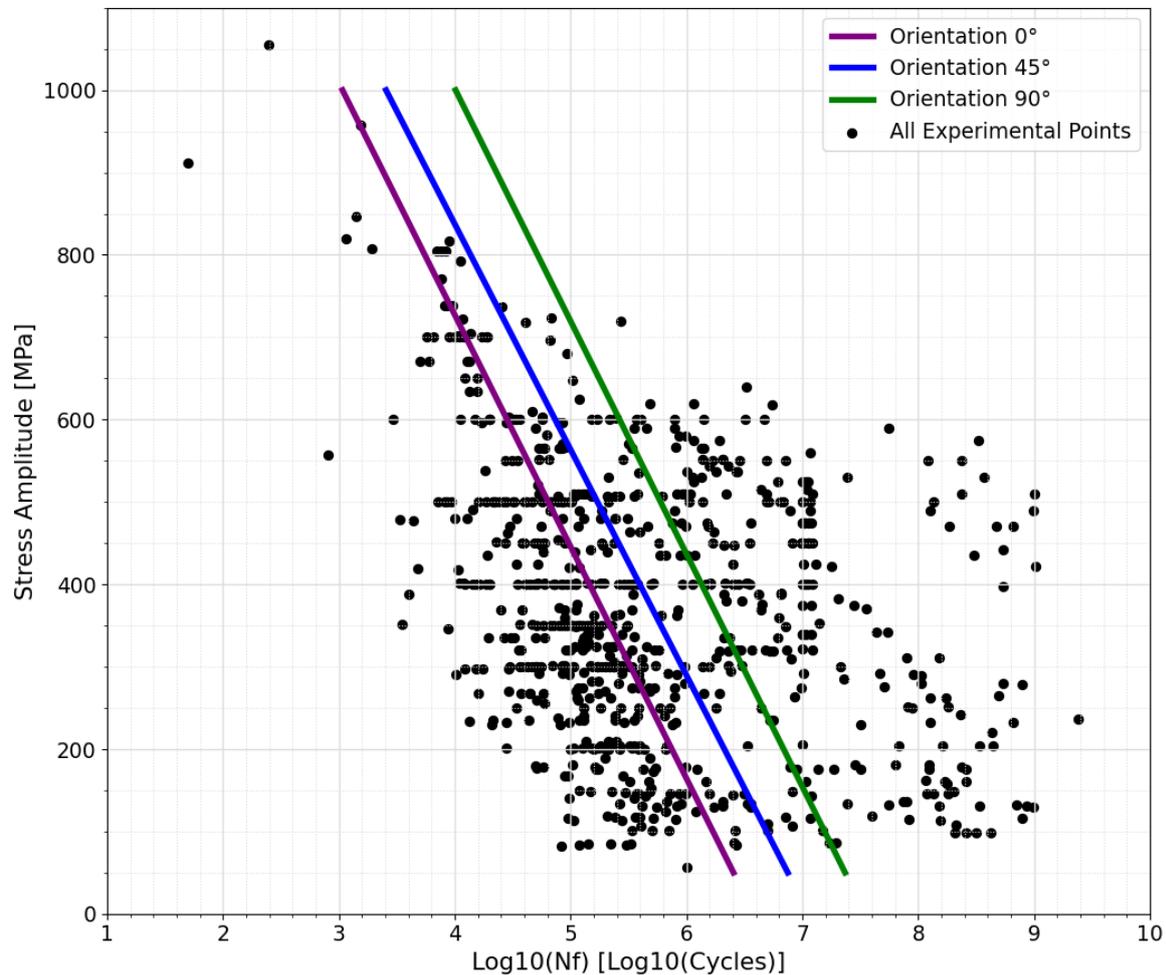


Figure 73, Varying the Building Orientation

### 13.2 Hatch Distance

This parameter varies from values of 0.05 mm to 0.25 mm and for this reason five values have been chosen: 0.05 mm, 0.10 mm, 0.15 mm, 0.20 mm and 0.25 mm. The building orientation is reset to 90°. Indeed, as we can see from *Figure 74*, low value of hatch distance gives better fatigue performances as opposed to high values. The lines are having their biases decreasing in an ordered manner from the 0.05 mm line to the 0.25 mm one. Recalling the role of the hatch distance (*Section 2*), the reason of this trend can be attributed to the fact that low value of hatch distance, allow the laser to better melt the powders, as we have a higher number of passes that are also closely packed, also giving the chance of having a partial remelt of the previous pass to grant a better bonding between the two subsequent passages. On the other hand, high values will lead to the contrary,

since the distance between each passage is too high to permit an optimal melting of the powders and a decent bonding with the previous passages.

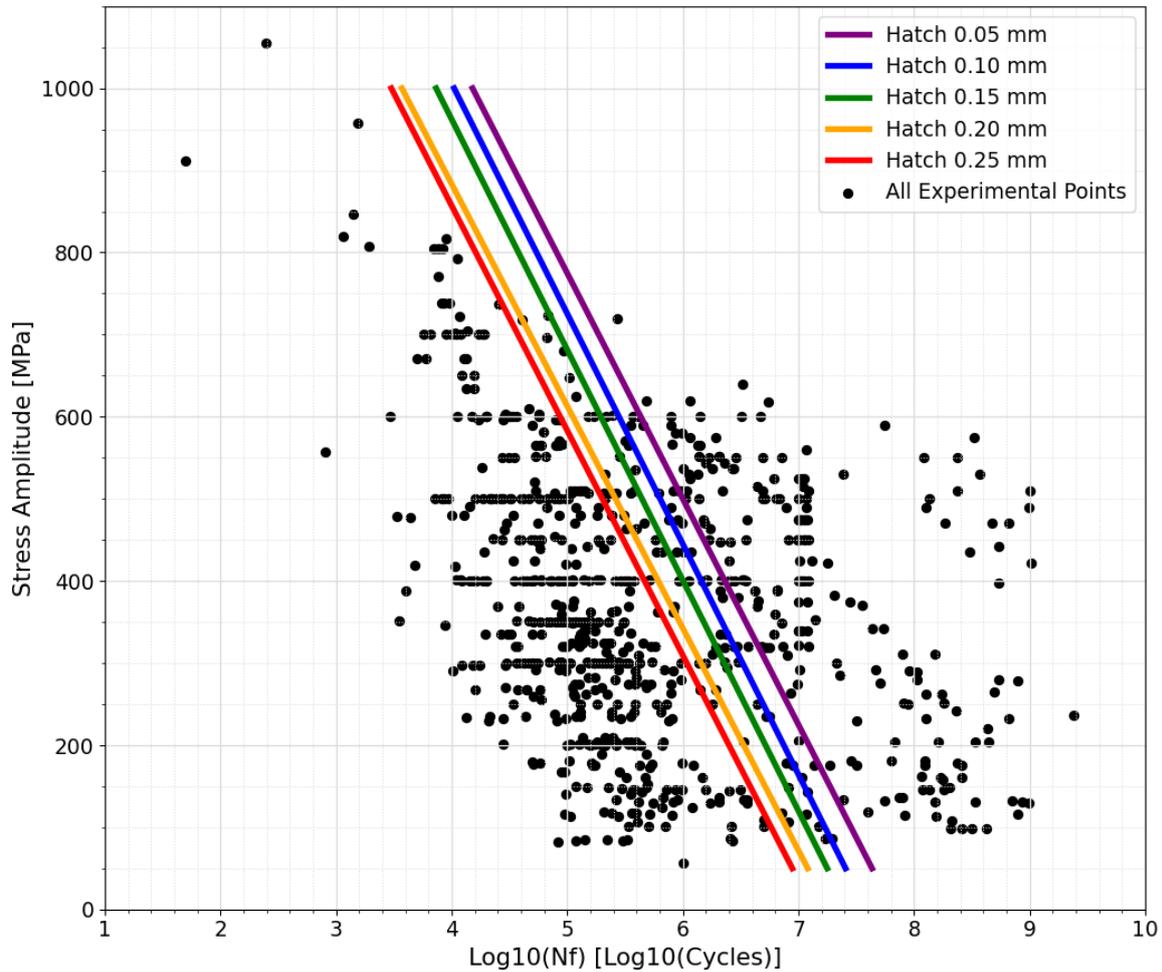


Figure 74, Varying the hatch distance

### 13.3 Layer Thickness

The layer thickness ranges from 30  $\mu\text{m}$  to 60  $\mu\text{m}$ . To be even more precise, we also have values of this parameter equal to 40  $\mu\text{m}$ , 45  $\mu\text{m}$  and 50  $\mu\text{m}$ , but the most frequent ones are 30  $\mu\text{m}$ , 45  $\mu\text{m}$  and 60  $\mu\text{m}$ , therefore those latter ones have been chosen. After resetting the hatch distance to the mean value of 0.12 mm, we can run the evaluation. In this case we cannot appreciate a significant difference between the three curves. The reason lies behind the fact that, as it was shown in *Section 6*, this parameter doesn't have a very high correlation to the fatigue life  $N_f$  (the coefficient was the lowest one with a value of 8%). To appreciate the

diversities, we have to perform a slight zoom of the same curve in the center region that ranges from 200 MPa to 600 MPa and from  $10^5$  to  $10^7$  cycles. We can see that 30  $\mu\text{m}$  leads to the worst prediction while 60  $\mu\text{m}$  gives us the best. Physically speaking the reason can be found in the fact that if we have small values of thickness, the laser disturbs the previously solidified layer while operating to create the next one, partially melting it, therefore not giving the new layer a solid surface to grasp on. On the other hand, if the layer is thick enough the contrary holds. Again, this difference is very subtle, therefore we must not worry too much about these variations.

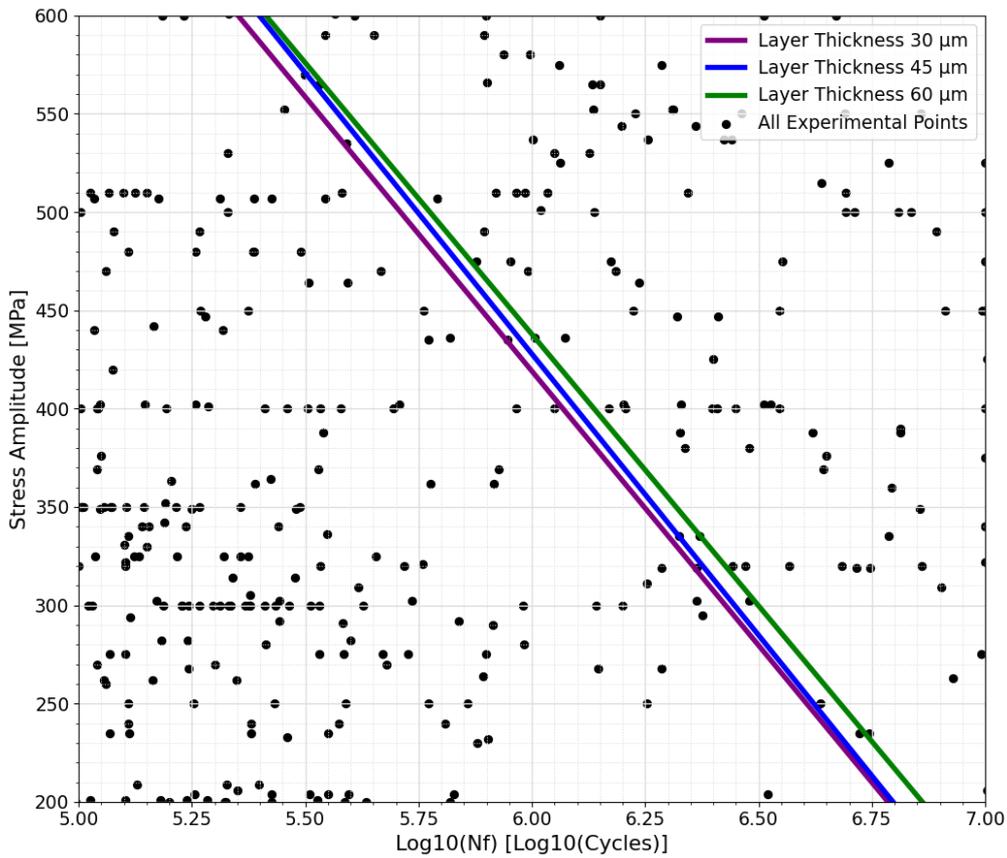
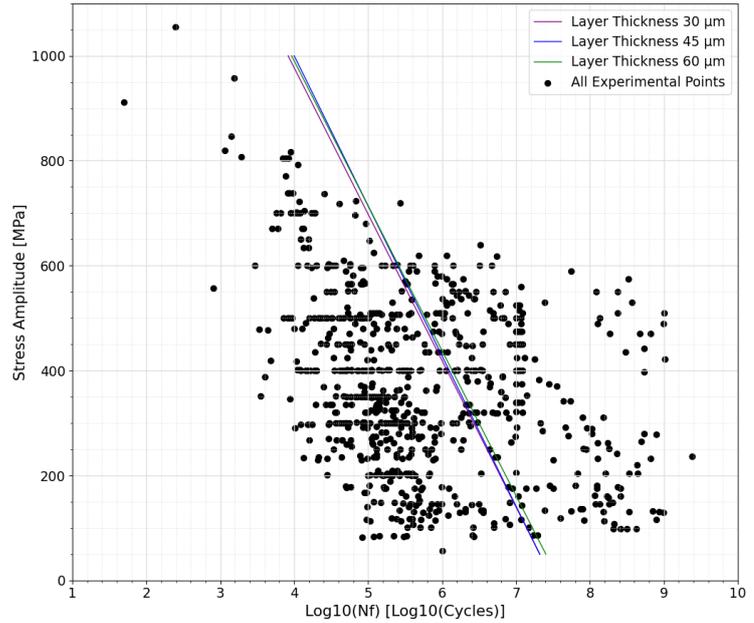


Figure 75, Varying the Layer Thickness

### 13.4 Speed/Power

Inside the database the cross-parameter speed/power ranges from values close to 0 mm/Ws to the maximum of 12.5 mm/Ws. Therefore, it has been chosen to have five values to investigate: 2.5, 5, 7.5, 10 and 12.5 mm/Ws. Indeed, if we reset the previous parameter to their mean values and we start producing the curves, we can see the following trend depicted in *Figure 76*.

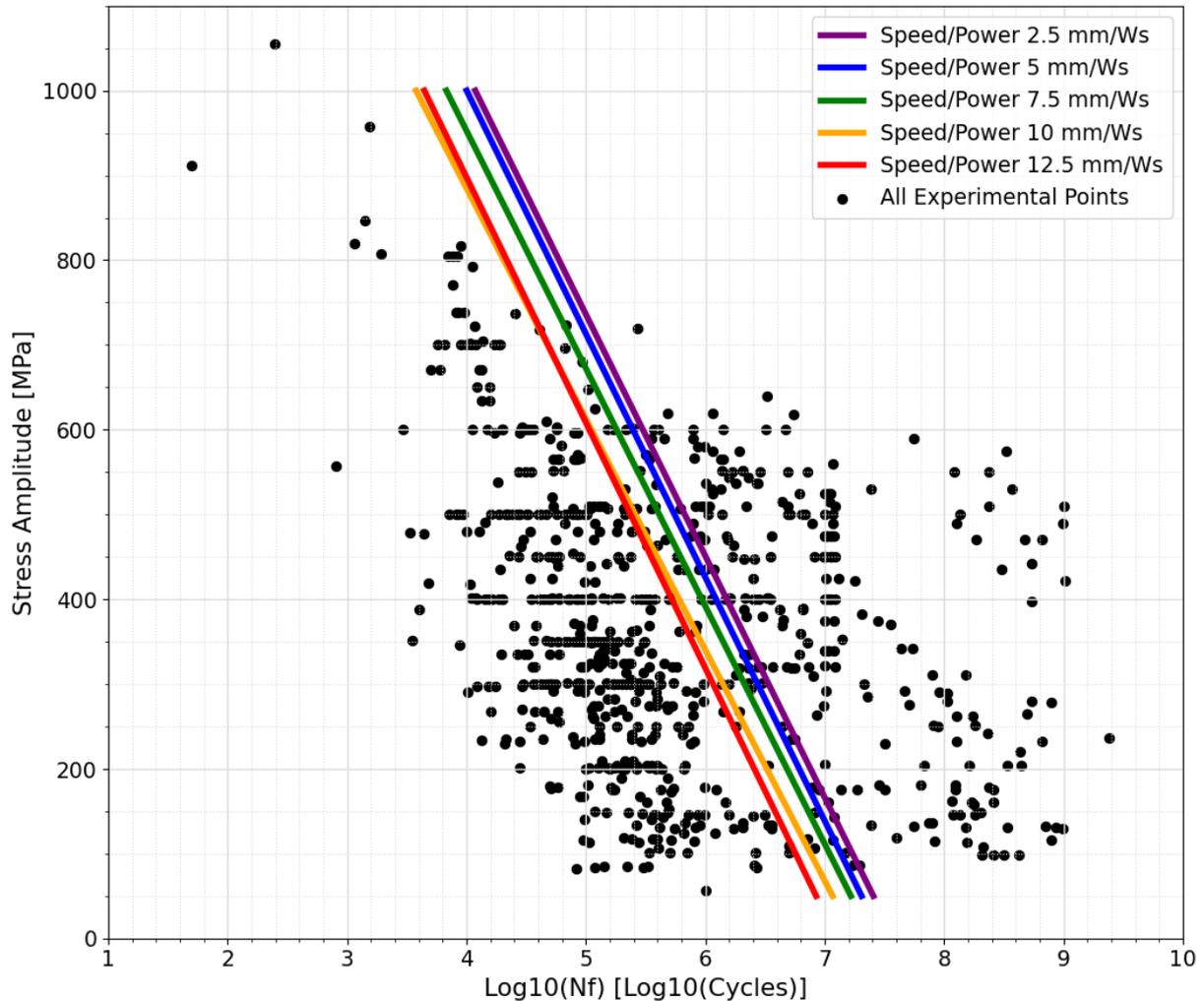


Figure 76, Varying the Speed/Power cross-parameter

We can see that low values of speed/power produce better fatigue performances with respect to high values of this parameter. The physical reason for this trend is simple. Recalling the laser power, we have that sufficiently high values should allow a good melting of the powders, while low values don't deliver enough energy to permit the bonding of the powders. On the other end, when it comes to the laser scan speed, if the laser is traveling too quickly, the material doesn't have sufficient time to create the solidified structure; conversely if the speed is moderate the powders can have sufficient time to bind with each other. Therefore, it's trivial to understand that if we have moderate speeds and relatively high powers (low speed/power, e.g. 2.5 mm/Ws) the quality of the finished part could

likely be better with respect to parts manufactured with high speeds and low powers (high speed/power, e.g. 12.5 mm/Ws). In fact, in the previously shown figure, we can see that low values of this parameter produce a better fatigue response with respect to high values.

### 13.5 Conclusions

After having understood which value of each process parameter leads to the best curve, we can understand which is the best combination that gives us the highest fatigue curve, therefore we can create a plot in which we evaluate the best curve by combining the best process parameters that we have found in the previous studies, again with the linear simplified neural network structure. The set of features fed to the neural network to evaluate it, is the same as before, but the following process parameters have been chosen.

Speed/Power	2.5 mm/Ws	Layer Thickness	60 $\mu\text{m}$
Building Orientation	90°	Hatch Distance	0.05 mm

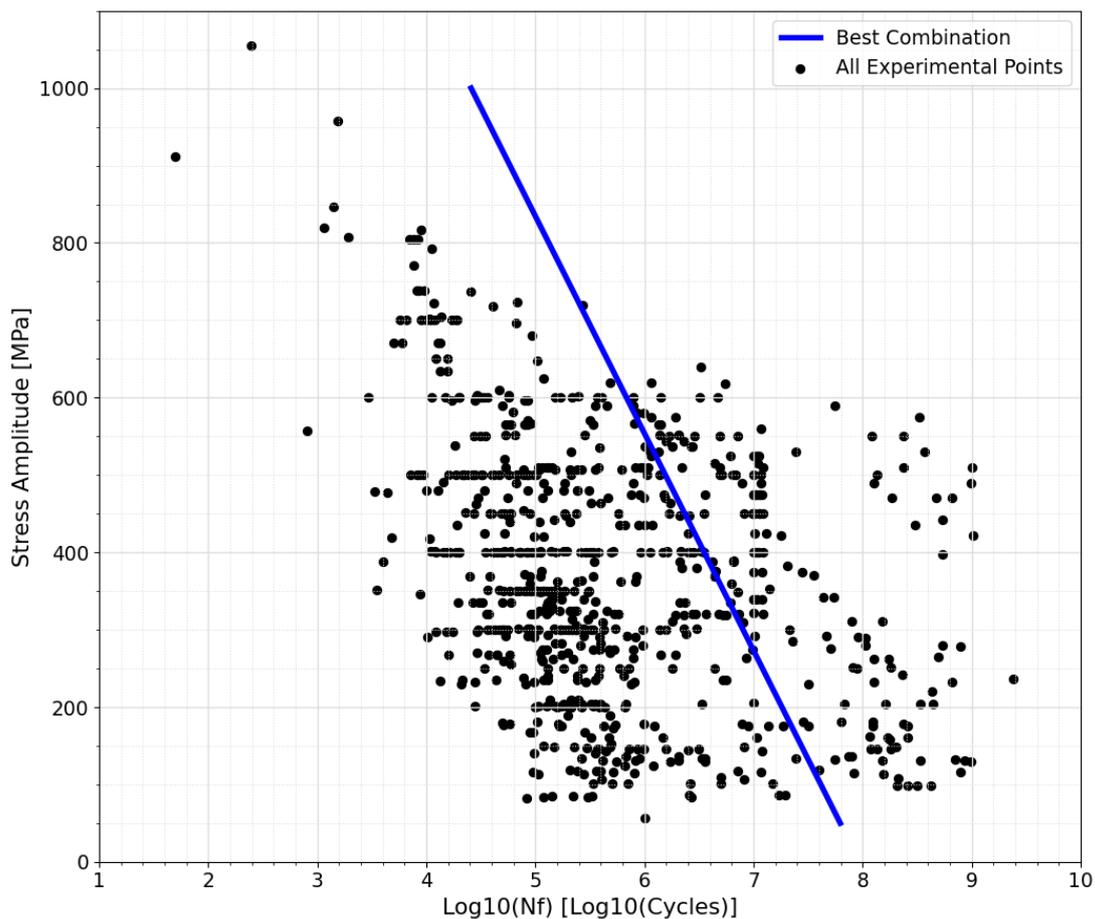


Figure 77, Best combination of process parameters

We can appreciate how this creates a curve that is greatly in the right part of the stress life diagram (Figure 77). However, we can proceed with further

investigations, as the Hot Isostatic Pressing thermal treatment hasn't been considered yet. In fact, we can also add the H.I.P. thermal treatment as it is known to improve fatigue properties. We can also compare the results produced before with the best combinations of process parameters and the annealing with and without the H.I.P. treatment to see if the aforementioned benefit occurs. We can also consider the synergic effect of both the annealing and HIPing (*Figure 78*). In this case the only surface treatments that have been used are the surface machining and polishing, while the annealing temperature has been set to the mean value of 735 °C, if the treatment is performed, while it is set to 20 °C if not.

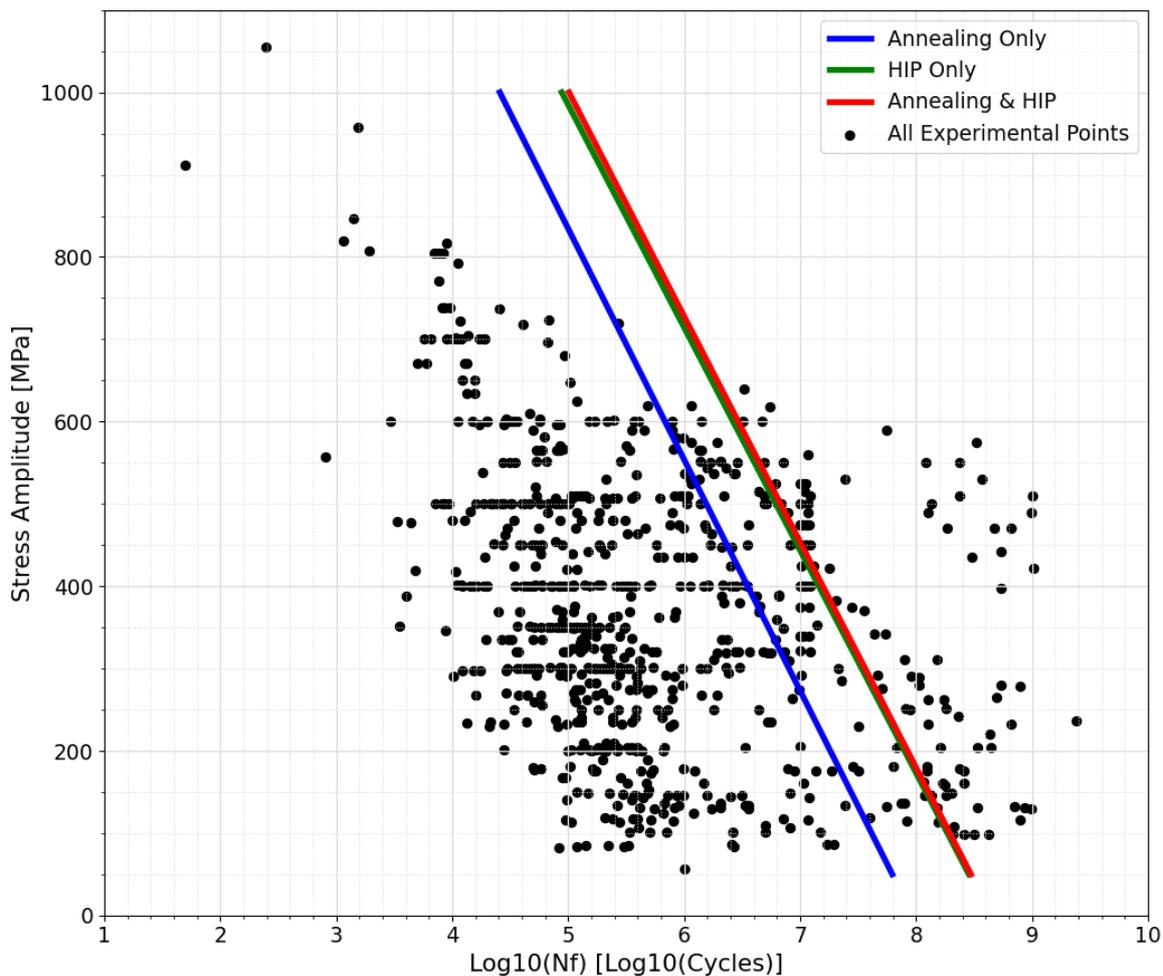


Figure 78, Best combination of process parameters with thermal treatments comparison

As we can expect, the HIP treatment gives a bigger contribution to the enhancement of the fatigue life with respect to the annealing treatment itself. Moreover, if we perform both treatments, first annealing and then HIPing, there is a slight improvement with respect to the HIP only.

With these evaluations, it is very easy to understand the power of neural networks. They can be basically used as a “program” in which we obtain a fatigue curve, after asking ourselves “what fatigue curve can I obtain with this combination of

process parameters and thermal and surface treatments?”. We simply input the desired set of features and we get the output curve without any significant expenditure of time and money, as we would normally do with traditional fatigue testing.

## 14. Defects Characterization Neural Network

While searching for the SN diagram fatigue experimental points in the articles, several information concerning the defect size and their distribution was found. It was decided to build another neural network, that is able to predict the defect size and their distribution depending on the process parameters. Unluckily, on the several articles, the information concerning the defect characterization was not continuous, as each researcher decided to use their own methodology to report the data. Therefore, a unique method to describe the defected distribution was needed. To unify the methodology, it was decided to use only post fracture surface scans, as many articles reported the CT scan of the piece before the fatigue testing, while we are interested on the defect characterization after the fracture on the fracture surface itself. To report the defects distribution, the Gumbel plot was used (*Figure 79*).

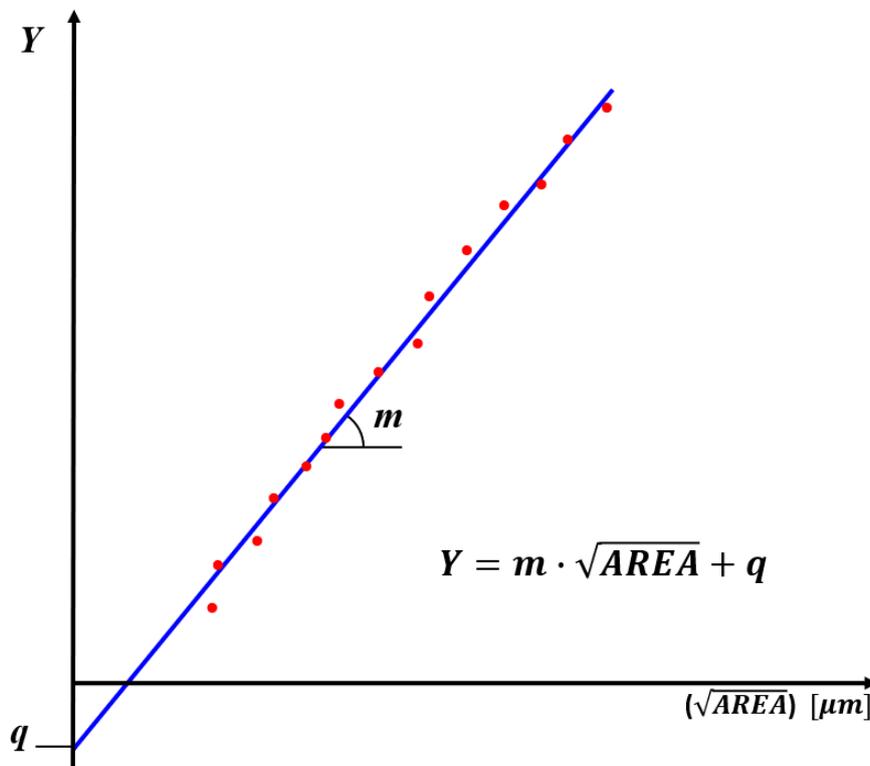


Figure 79, Gumbel Plot example

To produce a Gumbel plot, first a fracture surface needs to be investigated to characterize all the defects in terms of their size. Generally, the defect is approximated with a circumference, from which the equivalent diameter is evaluated. From this latter equivalent diameter, the parameter  $\sqrt{AREA}$  [ $\mu\text{m}$ ] is calculated with trivial formulas. Then the defects are ordered ascending from the smallest to the biggest in terms of  $\sqrt{AREA}$ . Once this is performed, the cumulative probability can be easily found by the following formula:

$$P_i = \frac{i}{N + 1}$$

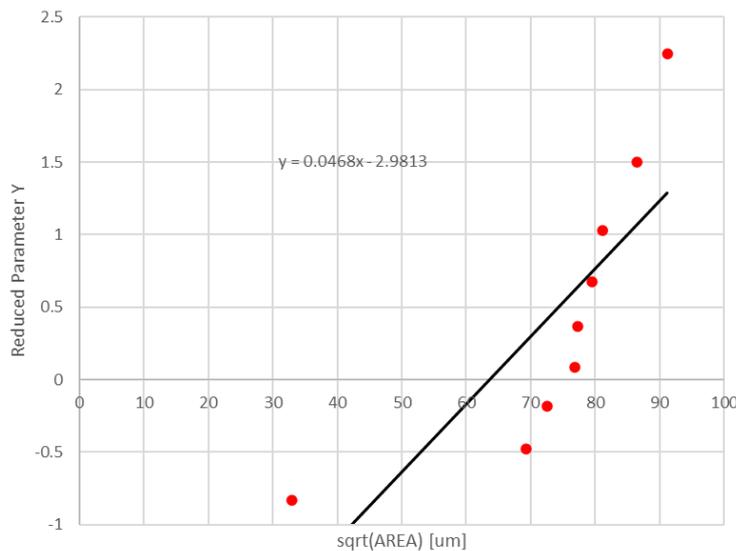
Where  $P_i$  is the cumulate probability of the  $i$ -th defect, and  $N$  is the total number of defects. Subsequently, the so-called *reduced parameter*  $Y$  is computed as follows to be the ordinate of the Gumbel plot:

$$Y_i = -\ln (-\ln(P_i))$$

If the procedure is carried out correctly, we will find that the Gumbel plot has its points that are always increasing. Lastly a linear fit can be produced to find the slope  $m$  and the bias  $q$ . To better understand the procedure, we can report as an example the article “Alegre2022”.

	$\sqrt{AREA} [\mu m]$	$\sqrt{AREA} [\mu m]$	$P_i$	$Y_i$
1	91.2	32.9	0.1	-0.834032445
2	32.9	69.3	0.2	-0.475884995
3	81.1	72.6	0.3	-0.185626759
4	77.3	76.9	0.4	0.087421572
5	79.6	77.3	0.5	0.366512921
6	69.3	79.6	0.6	0.671726992
7	72.6	81.1	0.7	1.030930433
8	76.9	86.5	0.8	1.499939987
9	86.5	91.2	0.9	2.250367327

Table 17, Example procedure to obtain a Gumbel plot



The procedure has been repeated 10 times in total, one for each data set. This time, instead of having 76 datasets, we're having less of them. Then for each data set the process parameters have been associated. In this case we are also having the *powder size minimum* and *powder size maximum* as all the datasets that were analyzed reported these two process parameters. Once again, the feature cross was not produced as this is an entirely diverse neural network with respect to the fatigue one. *Table 18* reports the training database in its entirety.

	Orientation [°]	Power [W]	Hatch [mm]	Speed [mm/s]	Layer thickness [µm]	Powder size min [µm]	Powder size max [µm]	<i>m</i>	<i>q</i>
0	90	175	0.12	710	30	20	63	0.0171	-2.7227
1	90	280	0.14	1200	30	38	38	0.0628	-3.5253
2	0	175	0.12	775	30	20	63	0.11885	-1.7436
3	45	175	0.12	775	30	20	63	0.046554	-1.2541
4	90	175	0.12	775	30	20	63	0.064336	-1.5958
5	90	400	0.12	150	60	28	54	0.0468	-2.9813
6	90	170	0.1	1200	30	23	46	0.1973	-1.5272
7	90	280	0.14	1200	30	15	53	0.1008	-3.5901
8	90	280	0.14	1200	30	15	45	0.049	-2.0425
9	90	300	0.14	1400	30	38	38	0.1016	-4.5457

	Name	Article	Reference
0	noName	Gunther2017	[6]
1	noName	Hu2020	[7]
2	0	Morel2019	[24]
3	45	Morel2019	[24]
4	90	Morel2019	[24]
5	As Built	Alegre2022	[17]
6	AF test	Macallister2022	[18]
7	Coarse	Tehrani2021	[20]
8	Fine	Tehrani2021	[20]
9	noName	HuWu2020	[25]

Table 18, Defects Neural Network training database

In several cases, the Gumbel plot was already produced in the article. Nonetheless, to check the continuity between the method used by the researchers of the article and the aforementioned method, all the points of the Gumbel plot were digitized and extracted, and the procedure was performed anyways. For nearly all the articles, the results coincided, while for another couple this didn't happen. For example, the article “Morel2019”, had the values of the bias *q* positive, which is not something that is typical in Gumbel plots. Once the training database is built, the simple *feedforward neural network* can be tuned

accordingly with the same procedures that have been explained in the previous sections.

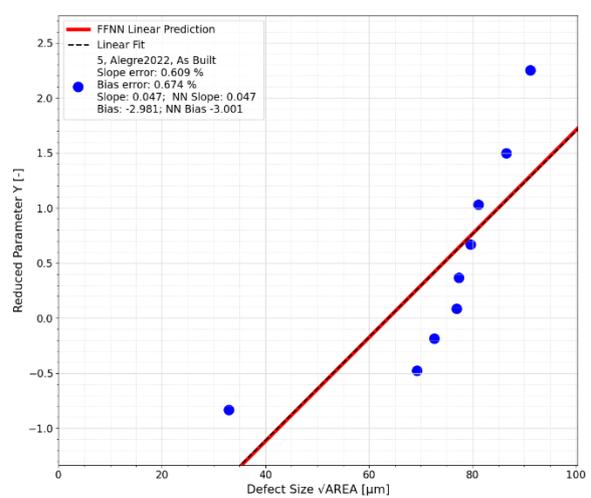
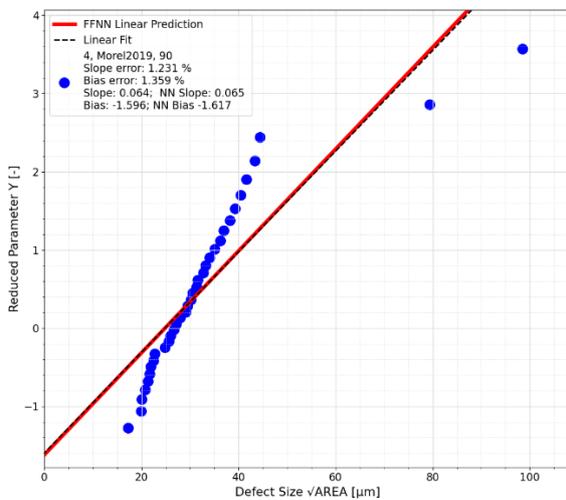
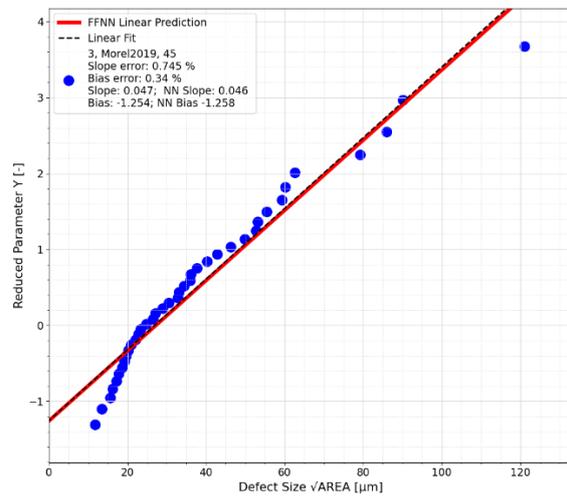
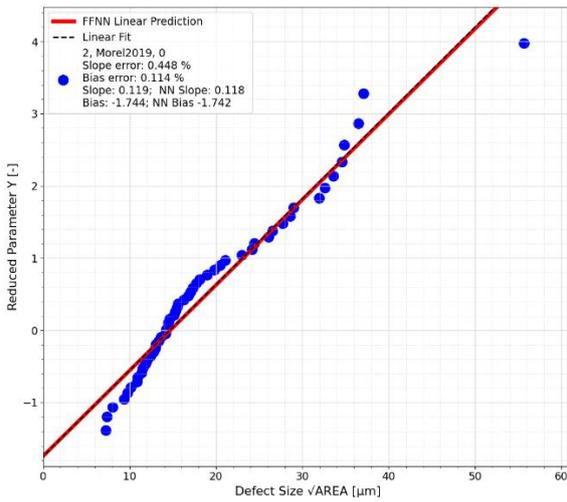
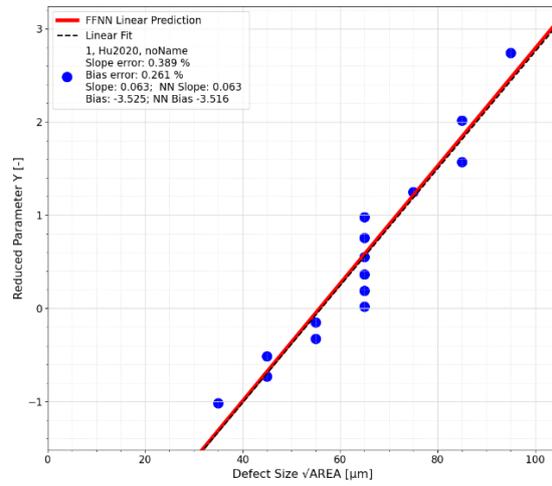
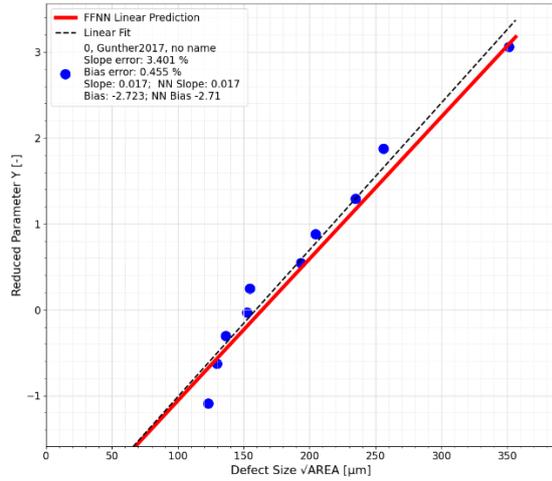
## 14.1 Neural network tuning and Results

After normalizing and scrambling the training database, the machine learning parameters and the neural network structure can be defined. The principles used in the four-parameter bilinear neural network are valid also here: since the shape of the final graph will be forced to be linear, we can use hyperbolic tangent layers to have a more efficient training, still being sure that we will not obtain curvilinear shapes. After numerous tests the machine learning parameters are the following: 500 epochs, learning rate = 0.001 and batch size half the size of the database, hence 5. The network has three *tanh* layers of 100, 75 and 50 neurons, with a linearly activated output layer with two nodes, as we need to predict two labels. Of course, since we have seven process parameters, the input layer has seven nodes. For this type of neural network, no thermal or surface treatment was considered, as there was not enough data to grant continuity in the database. Once the neural network has been trained, we can de-normalize the results of slope and bias after having fed the command “.predict” with the pertaining article, in the same exact way that has been presented in *Section 9*. Lastly a relative error concerning the slope  $m$  and the bias  $q$ , has been calculated for each dataset to be reported in the final figures with the real and predicted values of  $m$  and  $q$ .

$$\varepsilon_{rel,m} = \left| \frac{m_{real} - m_{N.N.}}{m_{real}} \right| \cdot 100$$

$$\varepsilon_{rel,q} = \left| \frac{q_{real} - q_{N.N.}}{q_{real}} \right| \cdot 100$$

We can finally report all the 10 figures to check the quality of the predictions.



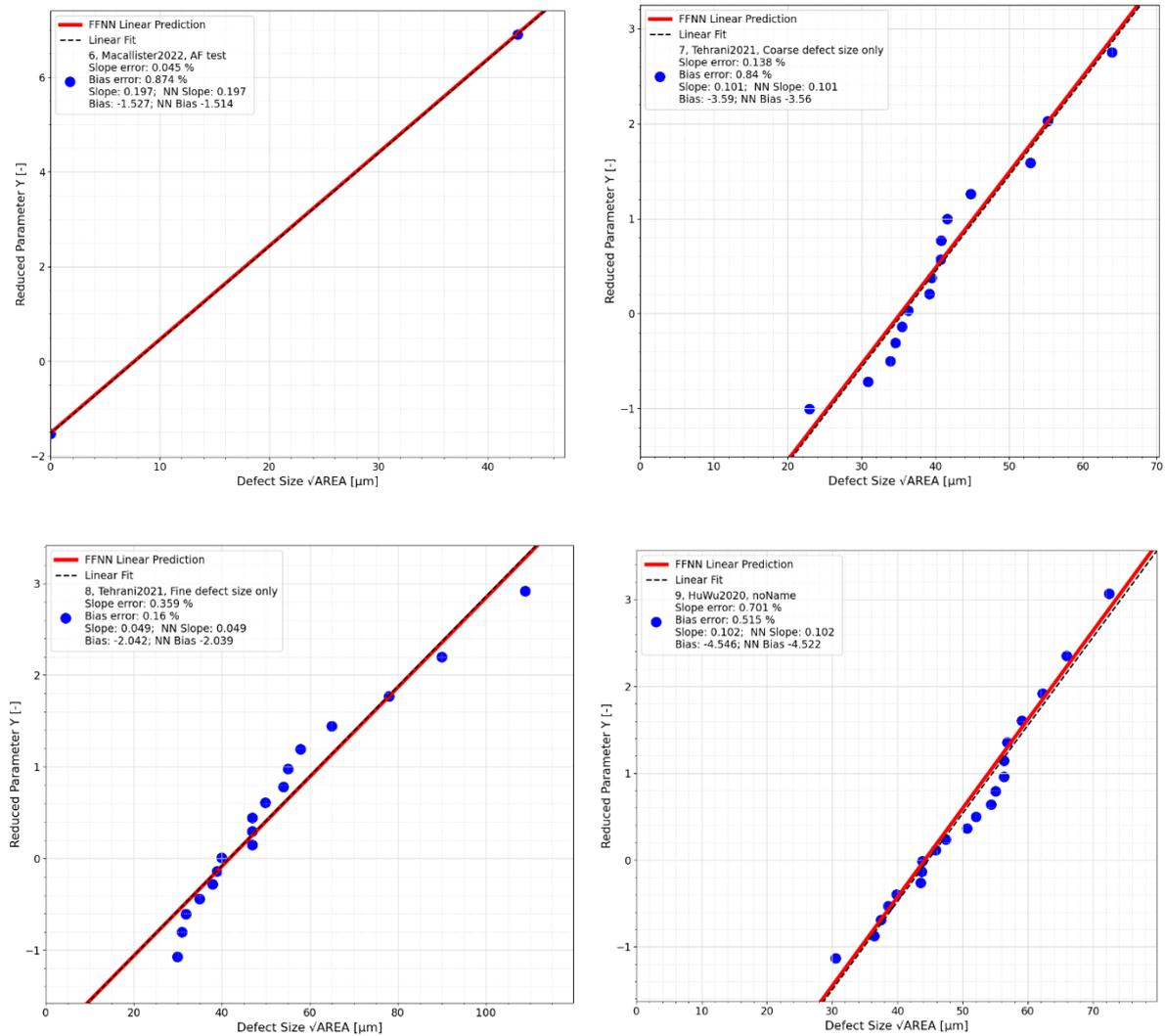


Figure 80, Evaluation of all the Gumbel plots

Indeed, as we can appreciate both graphically and numerically, the predictions are excellent. The red curves of the neural network prediction almost perfectly superimpose to the bilinear fits and the values of the relative errors are very small, as the real  $m$  and  $q$  are almost identical to the ones predicted by the neural network.

## References

- [1] [https://keep.lib.asu.edu/\\_flysystem/fedora/c7/233859/Sharma\\_asu\\_0010N\\_20221.pdf](https://keep.lib.asu.edu/_flysystem/fedora/c7/233859/Sharma_asu_0010N_20221.pdf)
- [2] <https://doi.org/10.1016/j.addma.2021.101876>
- [3] <https://doi.org/10.1016/j.ijfatigue.2022.107483>
- [4] <https://doi.org/10.1016/j.jcp.2018.10.045>
- [5] <https://doi.org/10.1111/ffe.13361>
- [6] <http://dx.doi.org/10.1016/j.ijfatigue.2016.05.018>
- [7] <https://doi.org/10.1016/j.matdes.2020.108708>
- [8] <http://dx.doi.org/10.1016/j.ijfatigue.2015.12.003>
- [9] <https://doi.org/10.1016/j.msea.2020.139385>
- [10] <http://dx.doi.org/10.1016/j.matdes.2015.12.135>
- [11] [10.3390/ma11040537](https://doi.org/10.1016/j.matdes.2015.12.135)
- [12] <https://doi.org/10.1016/j.optlastec.2021.107391>
- [13] <https://doi.org/10.1016/j.ijmecsci.2021.106591>
- [14] <https://doi.org/10.1016/j.matdes.2018.01.042>
- [15] [10.4028/www.scientific.net/AMR.816-817.134](https://doi.org/10.1016/j.matdes.2018.01.042)
- [16] <http://dx.doi.org/10.1016/j.matdes.2015.07.147>
- [17] <https://doi.org/10.1016/j.ijfatigue.2022.107097>
- [18] <https://doi.org/10.1016/j.actamat.2022.118189>
- [19] <https://iopscience.iop.org/article/10.1088/1757-899X/461/1/012052/meta>
- [20] <https://doi.org/10.1016/j.addma.2021.102584>
- [21] <https://doi.org/10.1080/21663831.2019.1609110>
- [22] <https://doi.org/10.1016/j.actamat.2020.05.041>
- [23] <https://doi.org/10.1016/j.ijfatigue.2020.106008>
- [24] <https://doi.org/10.1016/j.engfracmech.2019.03.048>
- [25] <https://doi.org/10.1016/j.ijfatigue.2020.105584>