***JULEN FERRO BAÑALES - HW3 - MACHINE LEARNING***

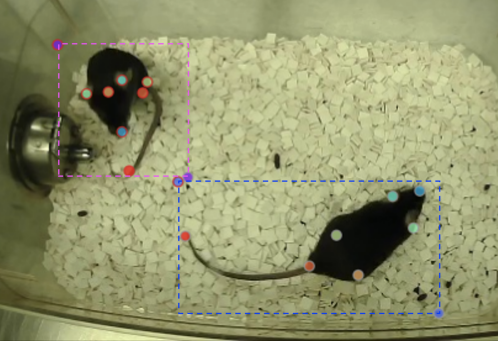
1. Introduction
   1. Project description
   2. Data analysis.
2. The structure Machine Learning model
   1. Neural Network
   2. Dependent model
   3. Independent model
3. Results of the simulations
   1. Overall performance
   2. Analysis of results.
4. Instructions
5. Conclusions
6. ***Introduction***

This report will explain how the author has developed a Machine Learning model able to predict the accurate location of a mouse in a 2D plane, based on the coordinates of different parts of its body such as: nose, left ear, right ear, left hip, right hip, tail base, tail end

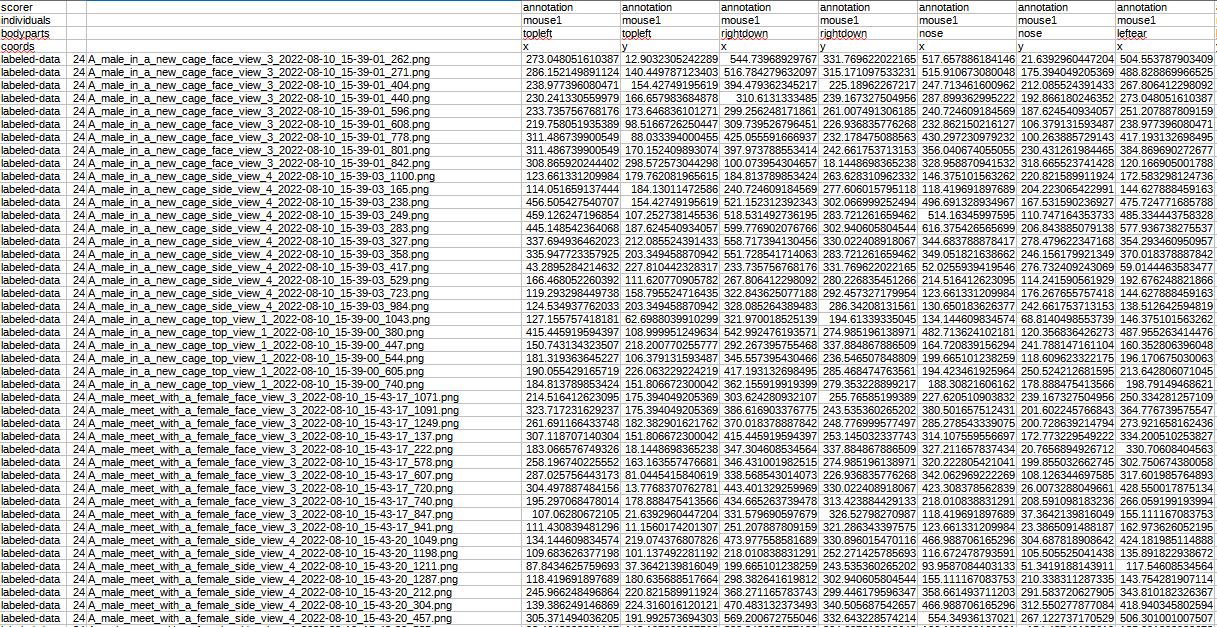
***1.1) Project description***

. This project deals with a Supervised Machine Learning problem, therefore, the labels must be obtained first in order to be able to have some feedback and compute the error. This is the point in which the author has realized some facts taught in class. For instance, the fact that the labeled data is really hard and expensive to get. The author has been given an annotation tool called DEEPLABCUT in order to work on some mice pictures and annotate the coordinates of their different body parts, and the coordinates of the smallest rectangle that encompasses each mouse.

As it has been explained, this first stage of the problem will lead to obtaining a .CSV file filled with different rows representing the data of each picture. Each row will be made up of different columns containing the values of the picture coordinates of the different parts of the mice’s body. As it can be seen in the figure below the annotations of the coordinates will be made and the DEEPLABCUT will fill one row of data per mouse with the coordinates of the annotations.

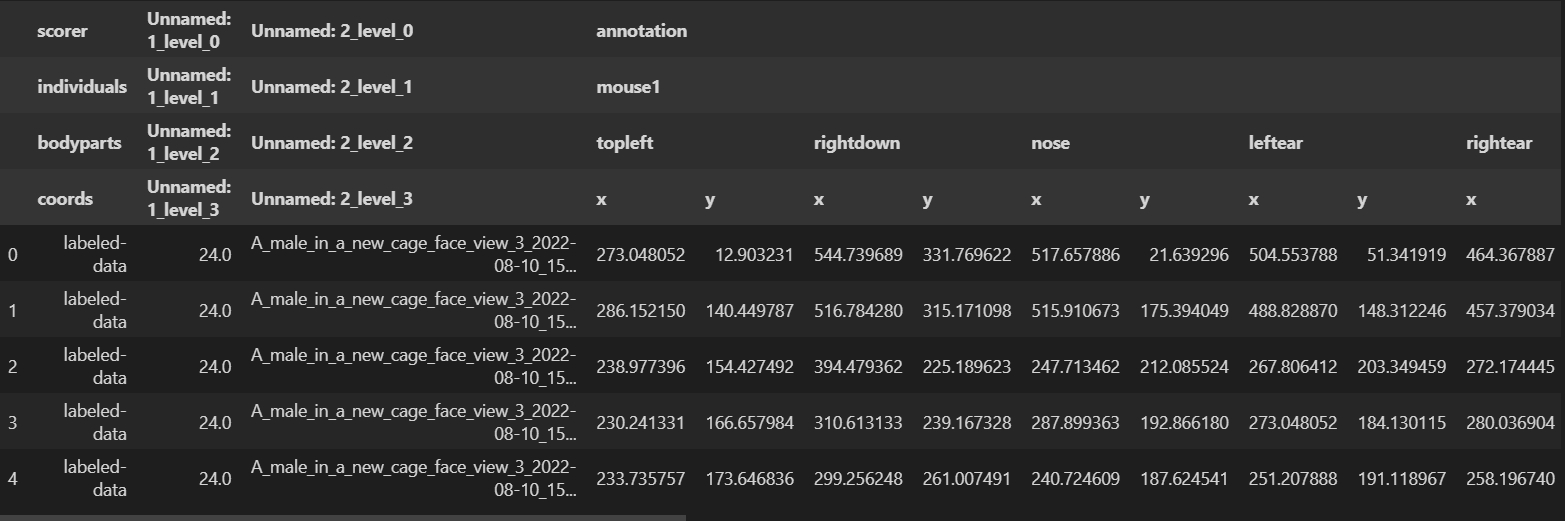


More precisely, the .CSV file with some of the annotations made, can be seen in the figure underneath. Each picture could contain more than one mouse, therefore the structure of the rows will not be constant and the software coded will have to be able to fix this problem.

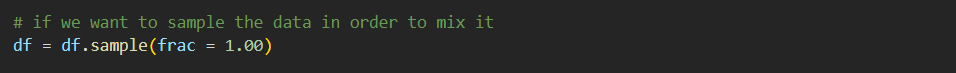


***1.2) Data analysis.***

First of all, the program will need to get the information of the .CSV file created by the DEEPLABCUT tool. In order to do so, the author has used the well-known pandas built-in function called *read\_csv(),* which returns the information in a dataframe made up of rows and columns *.* In the figure below, the raw data imported from the .CSV file can be seen.



Once the data is available in the program, the first steps have been to print some samples of the data in order to see which was the structure of it and to mix it with a *“.sample()”* method in order to get rid of the influence of the author’s ability to perform the annotations.See next figure.



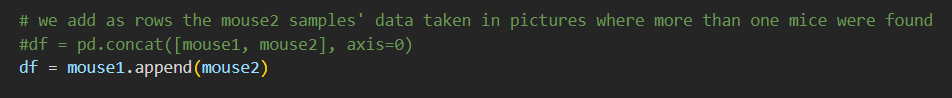
The author realized several issues to be taken into account. First of all, the labels of the column were made up of several layers, therefore the syntax for calling the columns of the pandas dataset would not be as easy as usual, when only one layer of the labels is used. Secondly, that the length of each row in terms of columns was not constant along the whole dataset. That is to say, as in some pictures, more than one mouse was found, and each row should represent a single sample, not a picture with more than one sample; the dataframe should be transformed and an additional row, per each row containing more than one mouse, should be created.

The author dealt with the first issue by reading python and pandas documentation and getting used to the proper syntax. Regarding the second issue, the data frame was splitted in two: one containing the mouse1 columns, and other one containing the mouse2 columns.

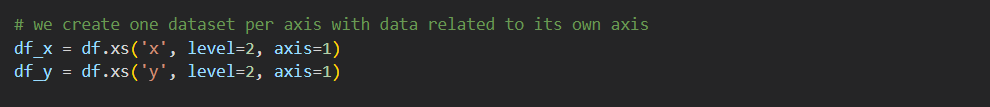
As when splitting the dataframe, some rows with NaN values were created, due to the fact that some rows were already filled with a single mouse’s data, not more than two. Therefore, the pandas *“dropna()”*  function with a threshold of one single NaN value was used in order to clean all those rows made up of any NaN value. See next figure.



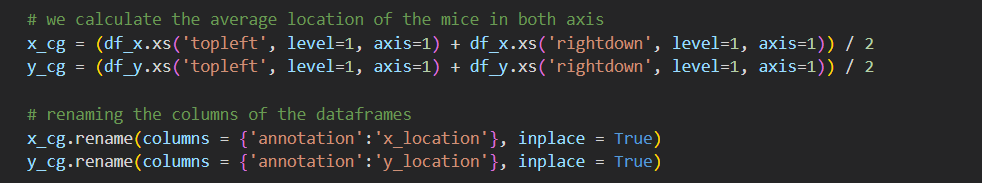
This second data frame was added or appended to the first one in order to get the *“df”* dataset, in which each row does represent a single mouse data. To do so, the *“.append”* method or the *“.concat”* method (with *“axis = 0”* for meaning that rows should be added, not columns) were used.



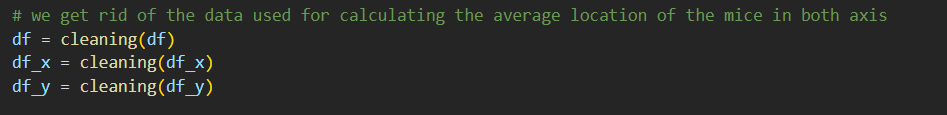
Then, the “df” was again splitted in two data frames each of them containing the information related to each axis (x or y). This way, the datasets that will be fed to the later mentioned “Dependent” and “Independent” models will be more readily accessible.



The next step was to calculate the average location of each mouse by computing the average of each coordinate for the *‘rightdown’* and *‘topleft’* columns. These values will be the final outputs of the designed models, so therefore they will be the values against which the results of the model will be checked in order to calculate the error.



Finally, as the *‘rightdown’* and *‘topleft’* data were already used and the average locations were added to the data frame, in order to foster an hygienic data frame and to get rid of redundant data, they will be eliminated from the dataset with a cleaning() function developed by the author.

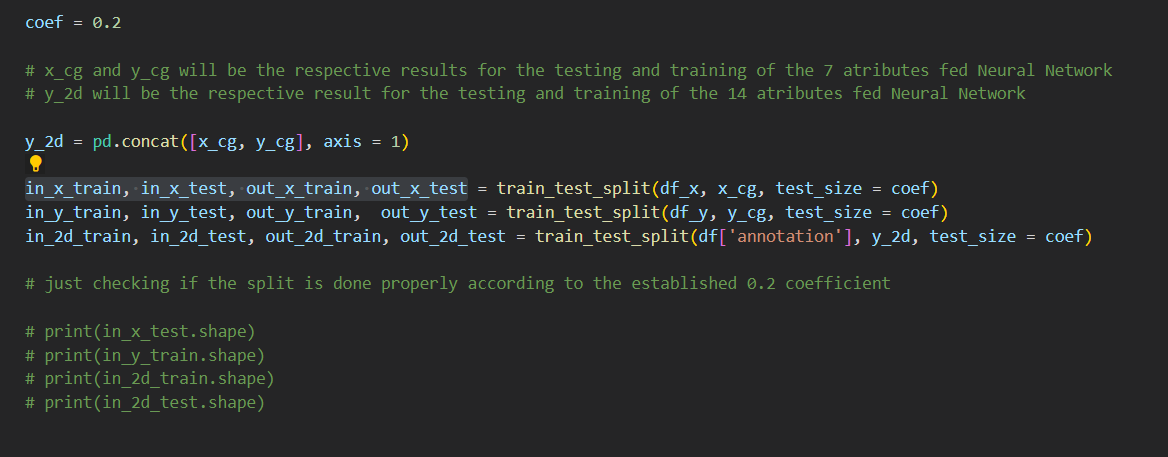


Once the data has been cleaned up, the author preceded to create the derivations of the overall datasets that will be fed into the model:

* in\_***XX***\_train
* in\_***XX***\_test
* out\_***XX***\_train
* out\_***XX***\_test

Being ***XX*** = [x, y, 2d].

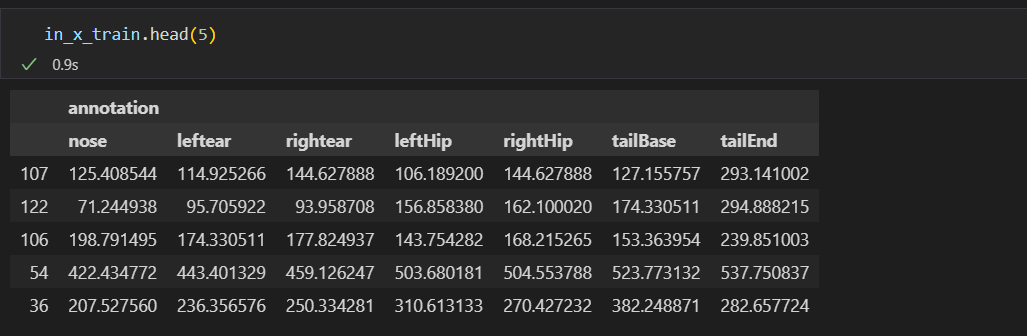
Therefore, 4 \* 3 = 12 new smaller data frames were created. The 2d one for *“The dependent model”*, and the x & y ones for the *“The independent model”.*



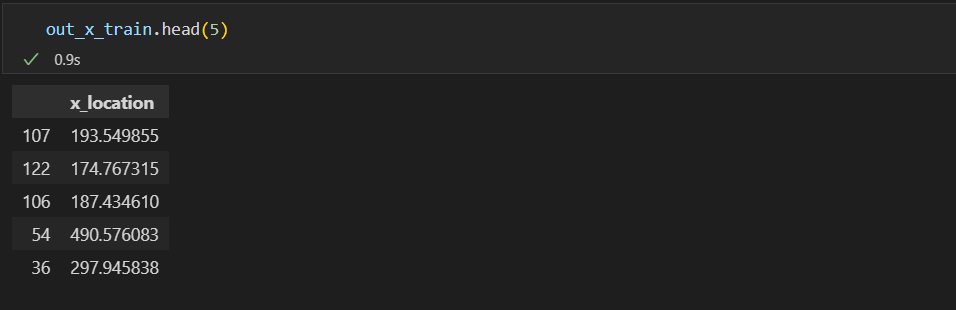
For the training and testing splitting, a 20% coefficient was used. Therefore, the testing dataframe’s length would be 20% of the original one and the rest of the rows would be for the training data frame.

Finally, in order to show some examples, four out of the twelve data frames that would be fed into the Neural Network are shown (*in\_x\_train, out\_x\_train, in\_2d\_train, out\_2d\_train*).

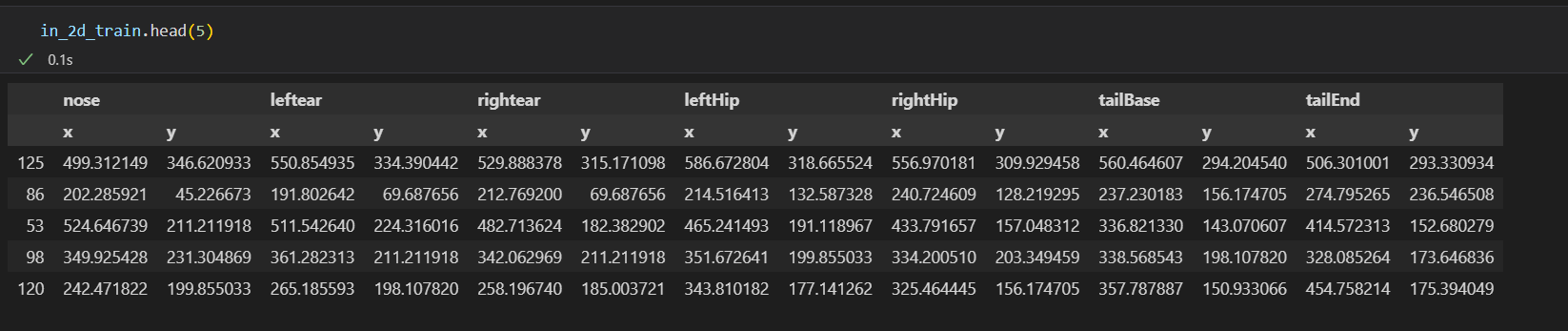
***in\_x\_train***



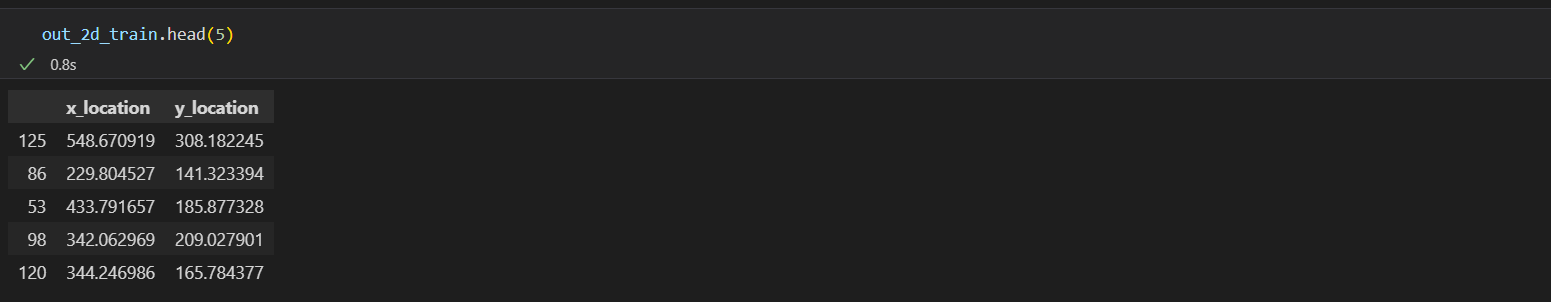
***out\_x\_train***

**

***in\_2d\_train***

**

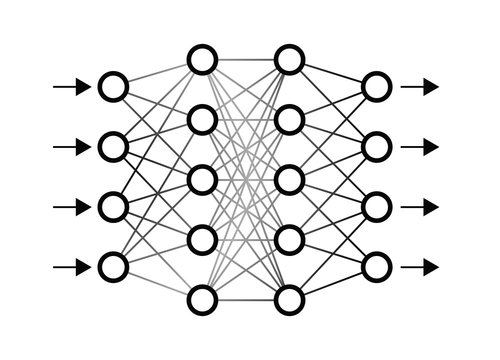
***out\_2d\_train***

**

1. ***The structure Machine Learning model***

***2.1) Feed Forward Neural Network***

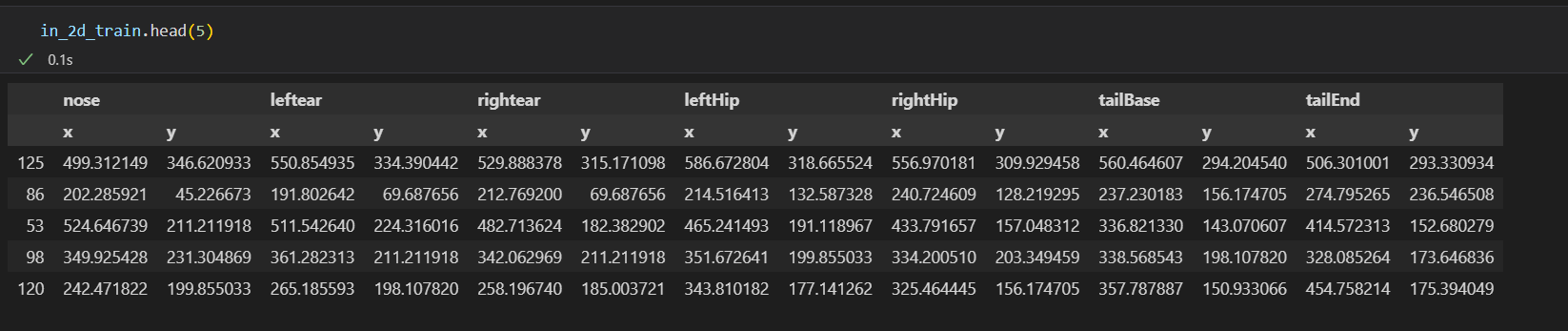
The model for tackling this Machine Learning project has been developed by using *“Feed Forward Neural Networks”.* The dimension of the input layer will vary depending on the simulation in question out of the two different ones developed by the author. Two hiddens layers of *“neurons”* equal number of neurons were used as it can be seen in the next figure. Forthe output layer it applies the same as for the input one, it will depend on the model chosen out of the two simulations designed.



***2.2) The dependent model***

*“The dependent model”* stands up for the idea that the y-axis values could have an influence on the x-axis results and vice versa. Therefore, this model or experiment would give room to any kind of dependence between x-axis and y-axis data.

In order to check that dependence by comparing this model to “The *independent model”*, as there are seven attributes per axis, and two axes, fourteen input data were fed into the neural network per row or per data sample.

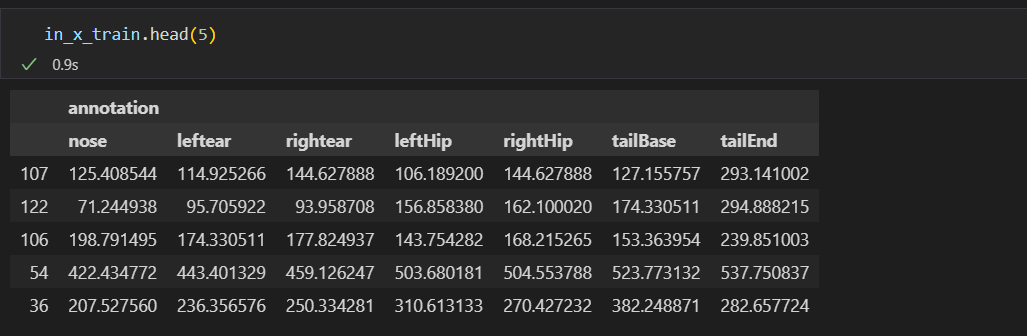
**

See in the figure above the fourteen attributes fed into the *“Feed Forward Neural Network”.*

***2.3) Independent model***

*“The independent model”* stands up for the idea that it would be impossible that the y-axis values could have an influence on the x-axis results or vice versa. Therefore, this model or experiment would never give room to any kind of dependence between x-axis and y-axis data because neither y-axis results, nor x-axis results were respectively calculated with x-axis and y-axis data.

In order to check the lack of dependencebetween the axis data, as there are seven attributes per axis, and only one axe per experiment, seven input data were fed into the neural network per row or per data sample. Nevertheless, two experiments were made, one for the x-axis and another one for the y-axis.



See in the figure above the seven attributes fed into the *“Feed Forward Neural Network”.*

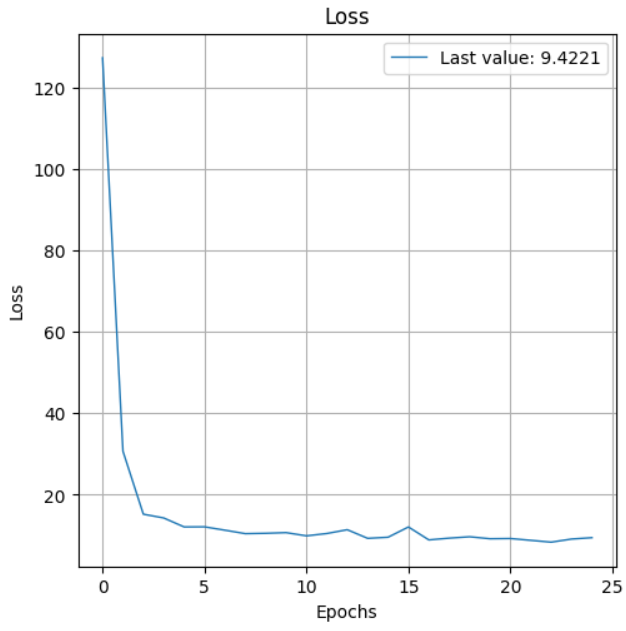
1. ***Results of the simulations***

This project has been based on three experiments made based on two different models.

***3.1) Overall performance***

The stopping criteria of the three experiments has been a threshold value for the Squared Error Loss as stated in the statement of the assignment. That is to say, the experiments would be running on an iterative loop, changing the *”Neural Network’s”* structure on each loop, until an acceptable error lower than the threshold is reached.

A Squared Error Loss of 2.0 has been established as a threshold to be reached. It must be highlighted that lower error values could be obtained as long as the computation time is increased, due to the fact that the *“Feed Forward Neural Network”* becomes more complex after each loop. Nevertheless, the author has determined that am2.0 value was enough in order to fulfill the accuracy and computing time criteria for the software. Taking into account that the coordinates data ranges from 0 to 500.0 more or less, a 2.0 value of threshold is pretty accurate. As it has been learnt from the report, there is a trade-off between the computing time and the accuracy of the algorithms. The chosen number of epochs was 25, more than enough taking into account the training error evolution over epochs seen in the figure below.



***3.2) Analysis of results.***

| Threshold = 2.0 | | | |
| --- | --- | --- | --- |
| The dependent model | | | |
|  | Neurons per layer | Computing time | Error |
| 2d-axis | 42 | 11.9 | 1.32 |
| The independent model | | | |
|  | Neurons per layer | Computing time | Error |
| x-axis experiment | 16 | 0.3 | 1.58 |
| y-axis experiment | 16 | 0.3 | 1.7 |

| Threshold = 1.0 | | | |
| --- | --- | --- | --- |
| The dependent model | | | |
|  | Neurons per layer | Computing time | Error |
| 2d-axis | 50 | 21.8 | 0.28 |
| The independent model | | | |
|  | Neurons per layer | Computing time | Error |
| x-axis experiment | 16 | 0.3 | 0.23 |
| y-axis experiment | 16 | 0.4 | 0.27 |

These have been the results obtained from the experiments. The three experiments have been launched two times. The first one , in the first table with an *“Squared Error Loss”* threshold of 2.0. And the second one, in the second table with an *“Squared Error Loss”* threshold of 1.0.

It can be seen that as long as the threshold is diminished, the computing time and the number of the needed neurons increases, mainly for the *“2d”* experiment*.* It can be seen that the single axis experiments were not really meaningful in terms of computing time due to the fact both experiments were really fast and do not give room to interpretation.

As it can be learnt, the more complex the model is, that is to say, the more neurons or more complex hidden layers it has, the more accurate relations between variables learned from the model during the training and the more accurate it behaves during the testing, thus leading to a smaller error.

In addition to this, it can be concluded that a hidden layer made up of more or less fifty neurons per layer, could be optimally enough in order to get accurate results and not waste time training more than needed by the “*Neural Network”*.

Finally, it can be seen that regarding error differences between the two different models, *“The dependent model”* and *“The independent model”*, error values for both models are more or less of the same order. That means that the axis data could be treated separately, thus leading to an independent system and giving room to a parallel distributed computing system. In this system, the computational time would be more or less the half of the one based on *“The dependent model”* due to the fact that two different and half-sized *“Neural Networks”* could be calculated at same time, one per axis.

1. ***Instructions***

The programming has been divided in chunks and has been made in order to make it the easiest way possible for the *Teaching Assistant* that will go through the report. Therefore, these are the unique steps to be followed.

* Make sure that the path of the .CSV file is properly written in the *“path*” variable
* Run the whole program
* Let it finish for 5 minutes more or less in the worst case\*

*\*The optimal structure (number of neurons) of the Neural Network will be obtained by running some simulations and training a lot of different models. It does not mean that the model is slow, it means that a lot of simulations will be carried out in a while loop until the error gets lower than a threshold in order to get an acceptable model.*

1. ***Conclusions***.

First of all, one of the main learnings that the author achieved during this project, was that Machine Learning experts really mean it when they say that the labeled data of the *Supervised Machine Learning* models is really expensive to obtain. In this project, the .CSV file needed in order to feed the data into the model, was not given. It was needed to do a lot of installation tasks and then, to obtain the .CSV file by using a great amount of pictures. The author really appreciated the efforts that must be done in order to get some clean and useful data.

Secondly, it also has to do with another statement frequently heard on *Machine Learning* lectures. Most of the *Data Scientist’s* time is spent on data cleaning and preparing activities. Building up a model and testing it are usually the last steps and the most easy ones, in terms of coding. However, for the data cleaning part, innumerable hindrances could be encountered starting from pandas library knowledge, going through the python syntax, and finishing with the great number of surprises that someone could encounter even in the simplest dataset.

Finally, it must also be said that even though the last part of the project, the one related to the model building, was the easiest one, it also was the one that the author enjoyed the most. Due to the fact that it requires a lot of statistics understanding and code debugging in order to reach the results desired. It also gives room to a lot of freedom and creativity when it comes to making up a *Feed Forward Neural Network* because innumerable combinations could appear. Nevertheless, the author has tried to reduce the contingencies of the model by making a couple of simple hidden layers in which the only unknown was the number of neurons per layer to be determined by the threshold established for the data error.

As it has been demonstrated, the more complex the model or the *Neural Network,* the more accurate results are obtained as long as the programmer is able to make the model work properly. That is to say, a better threshold could always be reached by adding more neurons, but that also means losing more time on the training stage of each model. Therefore,one of the concepts that must be learnt is that there will always be a trade-off between the computing time (and computing cost usually related to the computing time) and the accuracy. Also, it must be pointed out that the experiments double check the fact that the axis could be treated separately. That is to say, that x-axis data has no influence in y-axis data and vice versa. Therefore, the computing time could be reduced to the half by breaking down the model into two *“Neural Networks”,* one per axis, computing at same time like a parallel computing architecture or distributed computing systems.

In addition to this, the project has also been useful for getting to know the *“Conda”* environment as well as installing programs such as DEEPLABCUT from GitHub repositories.