JULEN FERRO BAÑALES - HW3 - MACHINE LEARNING

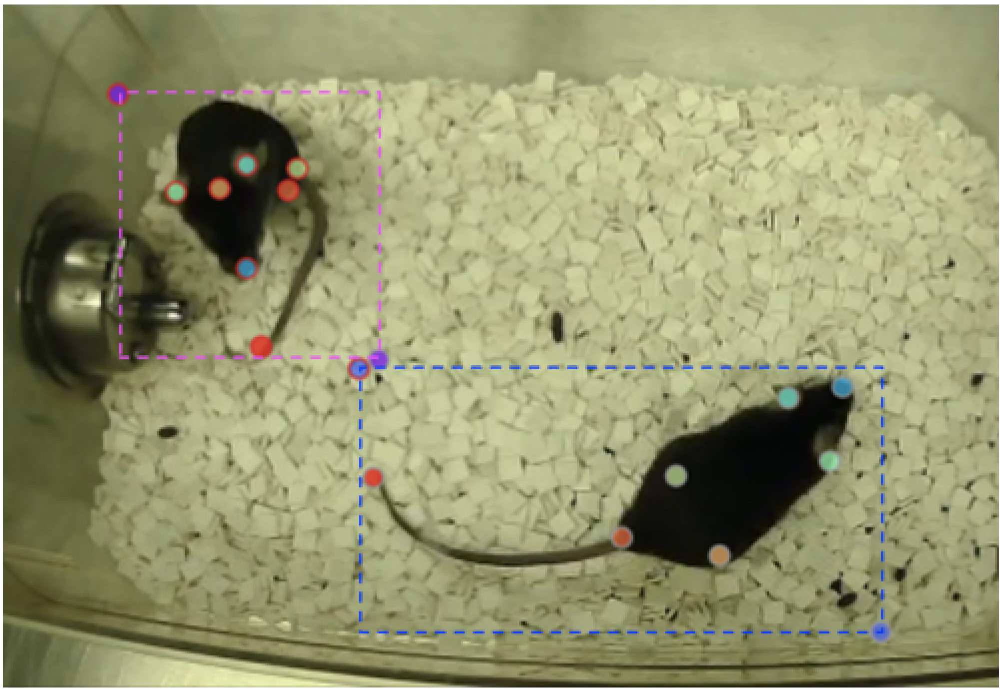
1) Introduction

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.

scorer			annotation	annotation	annotation	annotation	annotation	annotation	annotation
individuals			mouse1	mouse1	mouse1	mouse1	mouse1	mouse1	mouse1
bodyparts			topleft	topleft	rightdown	rightdown	nose	nose	leftear
coords			x	y	x	y	x	y	x
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_262.png	273.048051610387	12.9032305242289	544.73968929767	331.769622022165	517.657886184146	21.6392960447204	504.553787903409
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_271.png	286.152149891124	140.449787123403	516.784279632097	315.171097533231	515.910673080048	175.394049205369	488.828869966525
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_404.png	238.977396080471	154.42749195619	394.479362345217	225.18962267217	247.713461600962	212.085524391433	267.806412298092
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_440.png	230.241330559979	166.657983684878	310.6131333485	239.167327504956	287.899362995222	192.866180246352	273.048051610387
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_596.png	233.735756768176	173.646836101271	299.256248171861	261.007491306185	240.724609184569	187.624540934057	251.207887809159
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_608.png	219.758051935389	98.5166726250447	309.739526796451	226.936835776268	232.862150216127	106.379131593487	238.977396080471
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_778.png	311.486739900549	88.033394000455	425.055591666937	232.178475088563	430.297230979232	100.263885729143	417.193132698495
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_801.png	311.486739900549	170.152409893074	397.973788553414	242.661753713153	356.040674055055	230.431261984465	384.869690272677
labeled-data	24	A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_842.png	308.865920244402	298.572573044298	100.073954304657	18.1448698365238	328.958870941532	318.665523741428	120.166905001788
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_1100.png	123.661331209984	179.762081965615	184.813789853424	263.628310962332	146.375101563262	220.821589911924	172.583298124736
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_165.png	114.051659137444	184.13011472586	240.724609184569	277.606015795118	118.419691897689	204.223065422991	144.627888459163
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_238.png	456.505427540707	154.42749195619	521.152312392343	302.066999252494	496.691328934967	167.531590236927	475.724771685788
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_249.png	459.126247196854	107.252738145536	518.531492736195	283.721261659462	514.16345997595	110.747164353733	485.334443758328
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_283.png	445.148542364068	187.624540934057	599.776902076766	302.940605804544	616.375426565699	206.843885079138	577.936738275537
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_327.png	337.694936462023	212.085524391433	558.717394130456	330.022408918067	344.683788878417	278.479622347168	354.293460950957
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_358.png	335.947723357925	203.349458870942	551.728541714063	283.721261659462	349.051821638662	246.156179921349	370.018378887842
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_417.png	43.2895284214632	227.810442328317	233.735756768176	331.769622022165	52.0255939419546	276.732409243069	59.0144463583477
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_529.png	166.468052260392	111.620770905782	267.806412298092	280.226835451266	214.516412623095	114.241590561929	192.676248821866
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_723.png	119.293298449738	158.795524716435	322.843625077188	292.457327179954	123.661331209984	176.267655757418	144.627888459163
labeled-data	24	A_male_in_a_new_cage_side_view_4_2022-08-10_15-39-03_984.png	124.534937762033	203.349458870942	328.085264389483	286.34208131561	130.650183626377	242.661753713153	138.512642594819
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_1043.png	127.155757418181	62.6988039910299	321.970018525139	194.61339335045	134.144609834574	68.8140498553739	146.375101563262
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_380.png	415.445919594397	108.999951249634	542.992476193571	274.985196138971	482.713624102181	120.356836426273	487.955263414476
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_447.png	150.743134323507	218.200770255777	292.267395755468	337.884867886509	164.720839156294	241.788147161104	160.352806396048
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_544.png	181.319363645227	106.379131593487	345.557395430466	236.546507848809	199.665101238259	118.609623322175	196.170675030063
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_605.png	190.055429165719	226.063229224219	417.193132698495	285.468474763561	194.423461925964	250.524212681595	213.642806071045
labeled-data	24	A_male_in_a_new_cage_top_view_1_2022-08-10_15-39-00_740.png	184.813789853424	151.806672300042	362.155919919399	279.353228899217	188.30821606162	178.888475413566	198.79149468621
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_1071.png	214.516412623095	175.394049205369	303.624280932107	255.76585199389	227.620510903832	239.167327504956	250.334281257109
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_1091.png	323.717231629237	175.394049205369	386.616903376775	243.535360265202	380.501657512431	201.602245766843	364.776739575547
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_1249.png	261.691166433748	182.382901621762	370.018378887842	248.776999577497	285.278543339075	200.728639214794	273.921658162436
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_137.png	307.118707140304	151.806672300042	415.445919594397	253.145032337743	314.107559556697	172.773229549222	334.200510253827
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_222.png	183.066576749326	18.1448698365238	347.304608534564	337.884867886509	327.211657837434	20.7656894926712	330.70608404563
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_578.png	258.196740225552	163.163557476681	346.431001982515	274.985196138971	320.222805421041	199.855032662745	302.750674380058
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_607.png	287.025756443173	81.0445415840619	338.568543014073	226.936835776268	342.062969222269	108.126344697585	317.601985764893
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_720.png	304.497887484156	13.7768370762781	443.401329259969	330.022408918067	423.308378562839	26.0073288049661	428.550017875134
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_740.png	195.297068478014	178.888475413566	434.665263739478	313.423884429133	218.010838831291	208.591098183236	266.059199193994
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_847.png	107.06280672105	21.6392960447204	331.579690597679	326.52798270987	118.419691897689	37.3642139816049	155.11167083753
labeled-data	24	A_male_meet_with_a_female_face_view_3_2022-08-10_15-43-17_941.png	111.430839481296	11.1560174201307	251.207887809159	321.286343397575	123.661331209984	23.3865091488187	162.973626052195
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_1049.png	134.144609834574	219.074376807826	473.977558581689	330.896015470116	466.988706165296	304.687818908642	424.181985114888
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_1198.png	109.683626377198	101.137492281192	218.010838831291	252.271425785693	116.672478793591	105.505525041438	135.891822938672
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_1211.png	87.8434625759693	37.3642139816049	199.665101238259	243.535360265202	93.9587084403133	51.3419188143911	117.54608534564
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_1287.png	118.419691897689	180.635688517664	298.382641619812	302.940605804544	155.11167083753	210.338311287335	143.754281907114
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_212.png	245.966248496864	220.821589911924	368.271165783743	299.446179596347	358.661493711203	291.583720627905	343.810182326367
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_304.png	139.386249146869	224.316016120121	470.483132373493	340.505687542657	466.988706165296	312.550277877084	418.940345802594
labeled-data	24	A_male_meet_with_a_female_side_view_4_2022-08-10_15-43-20_457.png	305.371494036205	191.992573694303	569.200672755046	332.643228574214	554.34936137021	267.122737170529	506.301001007507

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.

	scorer	Unnamed: 1_level_0	Unnamed: 2_level_0	annotation								
	individuals	Unnamed: 1_level_1	Unnamed: 2_level_1	mouse1								
	bodyparts	Unnamed: 1_level_2	Unnamed: 2_level_2	topleft		rightdown		nose		leftear		rightear
	coords	Unnamed: 1_level_3	Unnamed: 2_level_3	x	y	x	y	x	y	x	y	x
0	labeled-data	24.0	A_male_in_a_new_cage_face_view_3_2022-08-10_15...	273.048052	12.903231	544.739689	331.769622	517.657886	21.639296	504.553788	51.341919	464.367887
1	labeled-data	24.0	A_male_in_a_new_cage_face_view_3_2022-08-10_15...	286.152150	140.449787	516.784280	315.171098	515.910673	175.394049	488.828870	148.312246	457.379034
2	labeled-data	24.0	A_male_in_a_new_cage_face_view_3_2022-08-10_15...	238.977396	154.427492	394.479362	225.189623	247.713462	212.085524	267.806412	203.349459	272.174445
3	labeled-data	24.0	A_male_in_a_new_cage_face_view_3_2022-08-10_15...	230.241331	166.657984	310.613133	239.167328	287.899363	192.866180	273.048052	184.130115	280.036904
4	labeled-data	24.0	A_male_in_a_new_cage_face_view_3_2022-08-10_15...	233.735757	173.646836	299.256248	261.007491	240.724609	187.624541	251.207888	191.118967	258.196740

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.

```
# if we want to sample the data in order to mix it
df = df.sample(frac = 1.00)
```

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.

```
mouse1 = df.xs('mouse1', level=1, axis=1)
mouse1 = mouse1.dropna(axis = 0)
mouse2 = df.xs('mouse2', level=1, axis=1)
mouse2 = mouse2.dropna(axis = 0)
```

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.

```
# we add as rows the mouse2 samples' data taken in pictures where more than one mice were found
#df = pd.concat([mouse1, mouse2], axis=0)
df = mouse1.append(mouse2)
```

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.

```
# we create one dataset per axis with data related to its own axis
df_x = df.xs('x', level=2, axis=1)
df_y = df.xs('y', level=2, axis=1)
```

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.

```
# we calculate the average location of the mice in both axis
x_cg = (df_x.xs('topleft', level=1, axis=1) + df_x.xs('rightdown', level=1, axis=1)) / 2
y_cg = (df_y.xs('topleft', level=1, axis=1) + df_y.xs('rightdown', level=1, axis=1)) / 2

# renaming the columns of the dataframes
x_cg.rename(columns = {'annotation':'x_location'}, inplace = True)
y_cg.rename(columns = {'annotation':'y_location'}, inplace = True)
```

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.

```
# we get rid of the data used for calculating the average location of the mice in both axis
df = cleaning(df)
df_x = cleaning(df_x)
df_y = cleaning(df_y)
```

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".*

```
coef = 0.2

# x_cg and y_cg will be the respective results for the testing and training of the 7 atributes fed Neural Network
# y_2d will be the respective result for the testing and training of the 14 atributes fed Neural Network

y_2d = pd.concat([x_cg, y_cg], axis = 1)

in_x_train, in_x_test, out_x_train, out_x_test = train_test_split(df_x, x_cg, test_size = coef)
in_y_train, in_y_test, out_y_train,  out_y_test = train_test_split(df_y, y_cg, test_size = coef)
in_2d_train, in_2d_test, out_2d_train, out_2d_test = train_test_split(df['annotation'], y_2d, test_size = coef)

# just checking if the split is done properly according to the established 0.2 coefficient

# print(in_x_test.shape)
# print(in_y_train.shape)
# print(in_2d_train.shape)
# print(in_2d_test.shape)
```

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

```
in_x_train.head(5)
```

✓ 0.9s

	annotation						
	nose	leftear	rightear	leftHip	rightHip	tailBase	tailEnd
107	125.408544	114.925266	144.627888	106.189200	144.627888	127.155757	293.141002
122	71.244938	95.705922	93.958708	156.858380	162.100020	174.330511	294.888215
106	198.791495	174.330511	177.824937	143.754282	168.215265	153.363954	239.851003
54	422.434772	443.401329	459.126247	503.680181	504.553788	523.773132	537.750837
36	207.527560	236.356576	250.334281	310.613133	270.427232	382.248871	282.657724

out_x_train

```
out_x_train.head(5)
```

✓ 0.9s

	x_location
107	193.549855
122	174.767315
106	187.434610
54	490.576083
36	297.945838

in_2d_train

```
in_2d_train.head(5)
```

✓ 0.1s

	nose		leftear		rightear		leftHip		rightHip		tailBase		tailEnd	
	x	y	x	y	x	y	x	y	x	y	x	y	x	y
125	499.312149	346.620933	550.854935	334.390442	529.888378	315.171098	586.672804	318.665524	556.970181	309.929458	560.464607	294.204540	506.301001	293.330934
86	202.285921	45.226673	191.802642	69.687656	212.769200	69.687656	214.516413	132.587328	240.724609	128.219295	237.230183	156.174705	274.795265	236.546508
53	524.646739	211.211918	511.542640	224.316016	482.713624	182.382902	465.241493	191.118967	433.791657	157.048312	336.821330	143.070607	414.572313	152.680279
98	349.925428	231.304869	361.282313	211.211918	342.062969	211.211918	351.672641	199.855033	334.200510	203.349459	338.568543	198.107820	328.085264	173.646836
120	242.471822	199.855033	265.185593	198.107820	258.196740	185.003721	343.810182	177.141262	325.464445	156.174705	357.787887	150.933066	454.758214	175.394049

out_2d_train

```
out_2d_train.head(5)
```

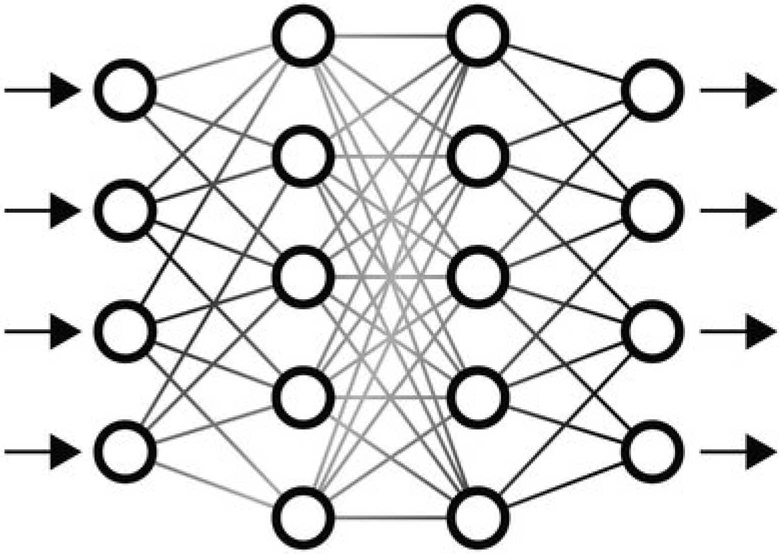
✓ 0.8s

	x_location	y_location
125	548.670919	308.182245
86	229.804527	141.323394
53	433.791657	185.877328
98	342.062969	209.027901
120	344.246986	165.784377

2) *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. For the 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.

```
in_2d_train.head(5)
```
✓ 0.1s

	nose		leftear		rightear		leftHip		rightHip		tailBase		tailEnd	
	x	y	x	y	x	y	x	y	x	y	x	y	x	y
125	499.312149	346.620933	550.854935	334.390442	529.888378	315.171098	586.672804	318.665524	556.970181	309.929458	560.464607	294.204540	506.301001	293.330934
86	202.285921	45.226673	191.802642	69.687656	212.769200	69.687656	214.516413	132.587328	240.724609	128.219295	237.230183	156.174705	274.795265	236.546508
53	524.646739	211.211918	511.542640	224.316016	482.713624	182.382902	465.241493	191.118967	433.791657	157.048312	336.821330	143.070607	414.572313	152.680279
98	349.925428	231.304869	361.282313	211.211918	342.062969	211.211918	351.672641	199.855033	334.200510	203.349459	338.568543	198.107820	328.085264	173.646836
120	242.471822	199.855033	265.185593	198.107820	258.196740	185.003721	343.810182	177.141262	325.464445	156.174705	357.787887	150.933066	454.758214	175.394049

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 dependence between 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.

```
in_x_train.head(5)
```
✓ 0.9s

	annotation						
	nose	leftear	rightear	leftHip	rightHip	tailBase	tailEnd
107	125.408544	114.925266	144.627888	106.189200	144.627888	127.155757	293.141002
122	71.244938	95.705922	93.958708	156.858380	162.100020	174.330511	294.888215
106	198.791495	174.330511	177.824937	143.754282	168.215265	153.363954	239.851003
54	422.434772	443.401329	459.126247	503.680181	504.553788	523.773132	537.750837
36	207.527560	236.356576	250.334281	310.613133	270.427232	382.248871	282.657724

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

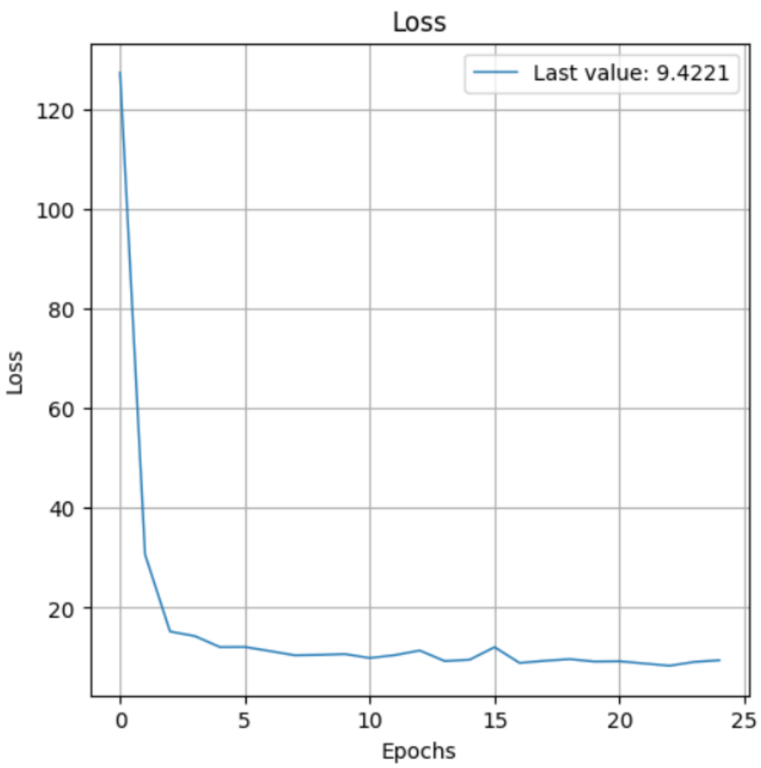
3) Results of the simulations

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.

4) 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.*

5) 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.