CSE-458
Soft Computing Project

Submitted to:
Prof: Amrita Chaturvedi

Submitted by:
Tushar Agarwal(19135110)
Rochak Bhardwaj(19095083)

Introduction

- We tried to predict whether the tumor is Benign or Malignant by looking at the shape, margin, density of the tumor and the age of the patient
- We are provided by a dataset that contains 6 columns and 961 rows. The columns are divided as shown in the table

Feature	Description	Value	Corresponding Value
AGE	Patient's Age		
SHAPE	Tumour Shape	1 2 3 4	Round Oval Lobular Irregular
MARGIN	Tumour Margin	1 2 3 4 5	Circumscribed Microlobulated Obscured Ill defined Spiculated
DENSITY	Density	1 2 3 4	High Constant Low Contains Fat
SEVERITY	Classification Of Tumor	0 1	Benign Malignant

Motivation

Cancer is a leading cause of death worldwide, accounting for nearly 10 million deaths in 2020, or nearly one in six deaths. For this reason, studies in this area have been extremely explored, proving to be a real challenge for researchers.

Currently, there are no highly specified mechanism for cancer detection Ultrasounds are a way of detecting abnormal patterns in tumors, thus raising the suspicion of cancer. Although ultrasounds have a relatively high efficiency, most of the time it is necessary to resort to biopsies to confirm if in fact the tumor is malignant or not.

Given the advances registered in the course of the last few decades in Artificial Intelligence, with special emphasis on Machine Learning, applications in this area have emerged for the stated problem. Thus, Machine Learning has been extremely useful to perform cancer diagnoses and it has already been proven that, with its help, the success rate increases significantly

Technology Used

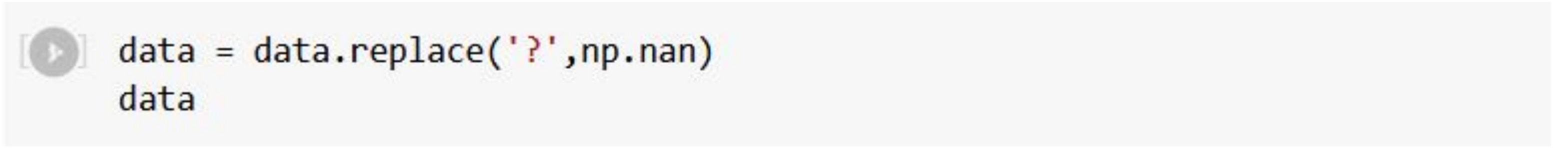
1. **Environment :** Python, jupyter notebook, google collab
2. **Data Processing and scientific computing :** Pandas, numpy
3. **Visualisation :** Matplotlib,seaborn
4. **Machine Learning library :** Scikit learn
5. **Deep Learning library :** Keras

Methodology

Analysis and treatment of null values

Now we will be analysing the NULL values.

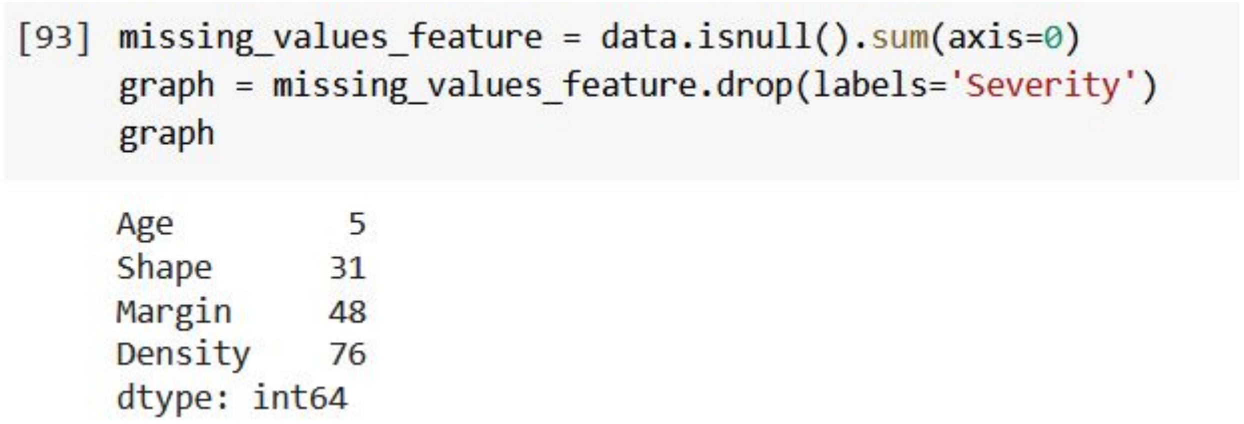
The null values is shown as ‘?’ in our dataset. We will convert all null values to NaN



```
data = data.replace('?',np.nan)
data
```

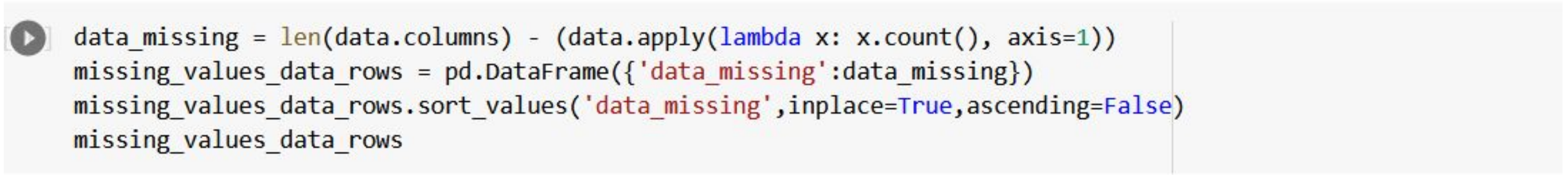
NULL values by feature

We try to find out the NULL values by feature to know which feature has the most number of missing values



```
[93] missing_values_feature = data.isnull().sum(axis=0)
     graph = missing_values_feature.drop(labels='Severity')
     graph

     Age          5
     Shape       31
     Margin      48
     Density     76
     dtype: int64
```

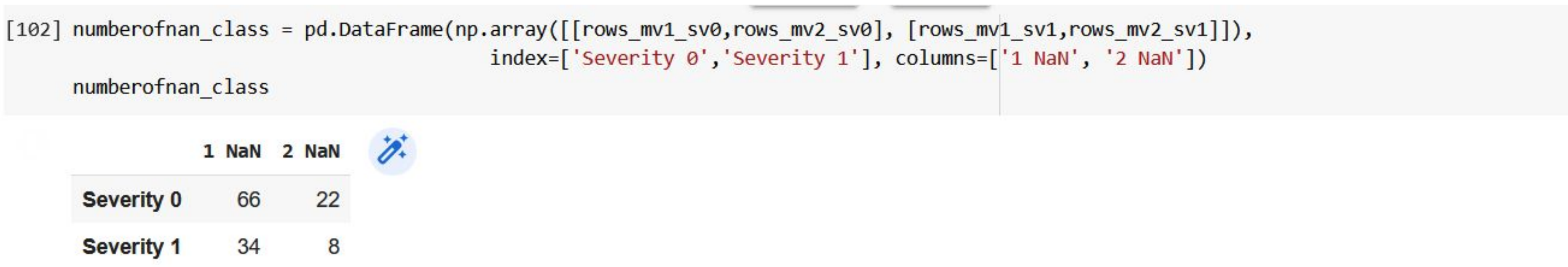
NULL values by line



```
data_missing = len(data.columns) - (data.apply(lambda x: x.count(), axis=1))
missing_values_data_rows = pd.DataFrame({'data_missing':data_missing})
missing_values_data_rows.sort_values('data_missing',inplace=True,ascending=False)
missing_values_data_rows
```

Through this we found out that the maximum number of NULL values per line is 2

NULL values by severity



```
[102] numberofnan_class = pd.DataFrame(np.array([[rows_mv1_sv0,rows_mv2_sv0], [rows_mv1_sv1,rows_mv2_sv1]]),
                                         index=['Severity 0','Severity 1'], columns=['1 NaN', '2 NaN'])
      numberofnan_class
```

```
            1 NaN  2 NaN
Severity 0     66     22
Severity 1     34      8
```

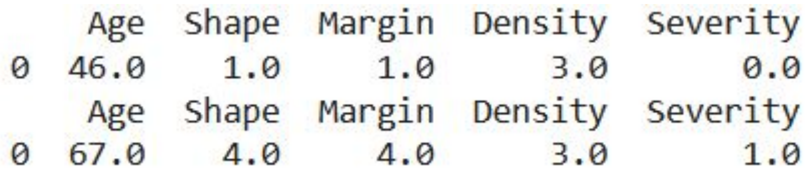
For a rating of 0, the values continue to
be extremely higher than for a rating of 1 for both 1 NaN and 2 NaN

Treatment of NULL values

After the analysis we conclude that the NULL values are not biased , this may seem inconsistent from our previous conclusions but this disparity is due to the existence of a greater number of samples for class 0 (516) than for class 1 (445).

Therefore we decide to replace the NaN of rows with one NaN value with the mode of that class and drop the rows with more than one NaN values

Mode of the classes are given below

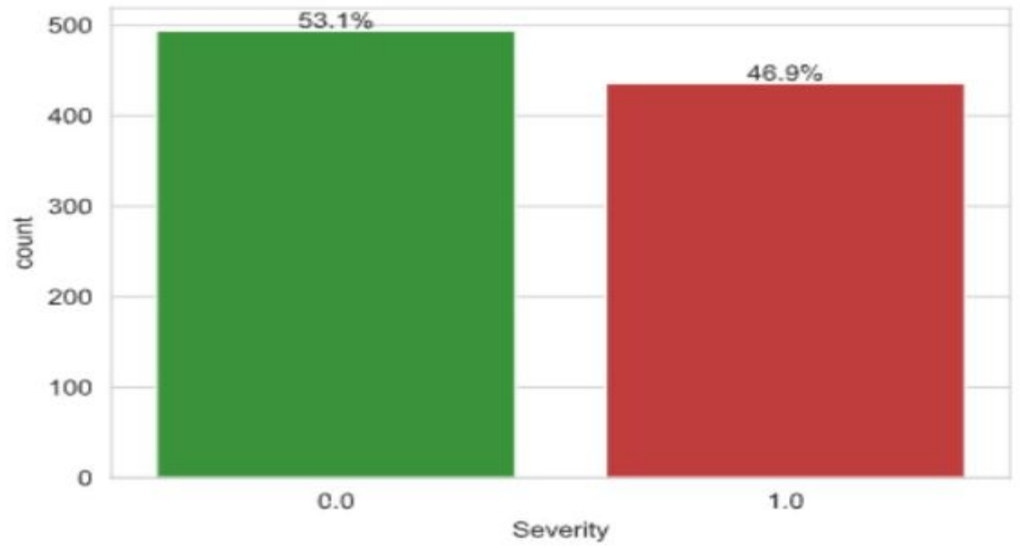


```
    Age  Shape  Margin  Density  Severity
0  46.0    1.0     1.0      3.0       0.0
    Age  Shape  Margin  Density  Severity
0  67.0    4.0     4.0      3.0       1.0
```

Exploratory Data Analysis (EDA)

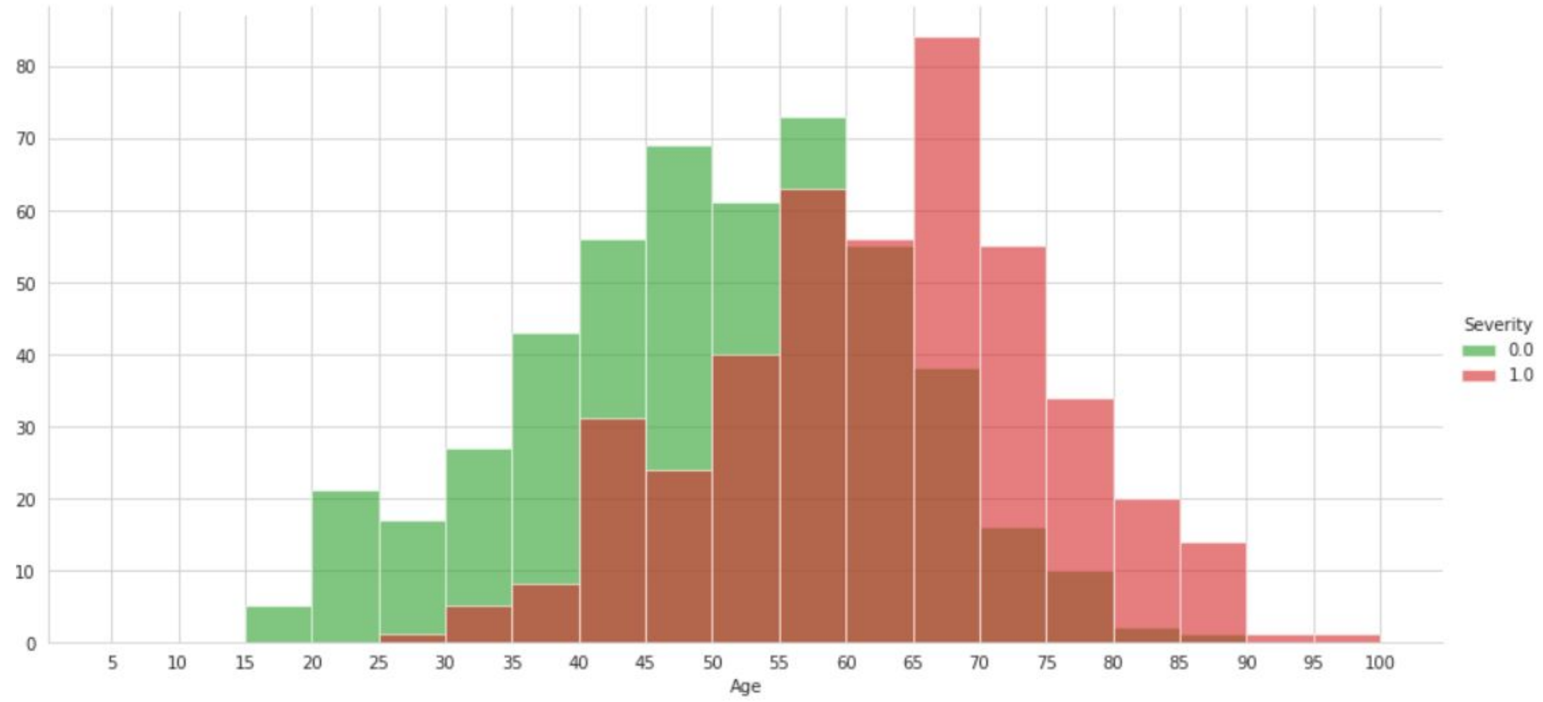
We start by evaluating the equilibrium of the dataset

Observing this graph, we can verify that the dataset is in fact quite balanced, presenting a total of 53.1% of samples for benign tumors and 46.9% for malignant tumors



Now we analyse age in general with the severity

Observing this graph we found older people have higher tendency to have malignant tumours

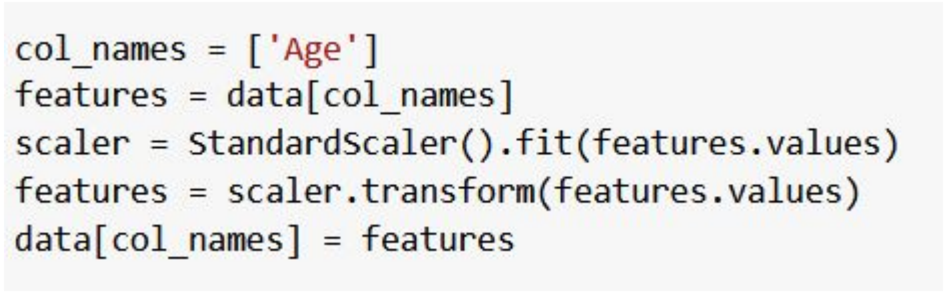


Data Processing

Data Processing for Numerical Data

For numerical data standardisation is a recommended preprocessing phase to improve the performance of neural networks

The numerical data present in the dataset is summarized by the Age feature. Due to the above reason, this feature was submitted to the standardization process, which was performed using the StandardScaler present in SkLearn



```
col_names = ['Age']
features = data[col_names]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
data[col_names] = features
```

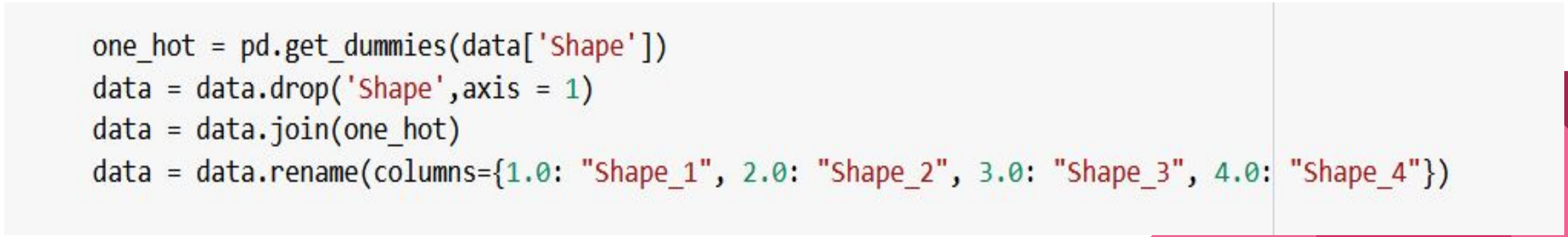
Data Processing for Categorical Data

For categorical data, One Hot Encoding is a recommended process to improve the performance of neural networks

Categorical data are variables that contain labeled values rather than numeric values. The number of possible values is usually limited to a fixed set and each value represents a different category.

The categorically data of the dataset that was submitted to the one hot encoding process were the following features

1. Margin
2. Shape
3. Density

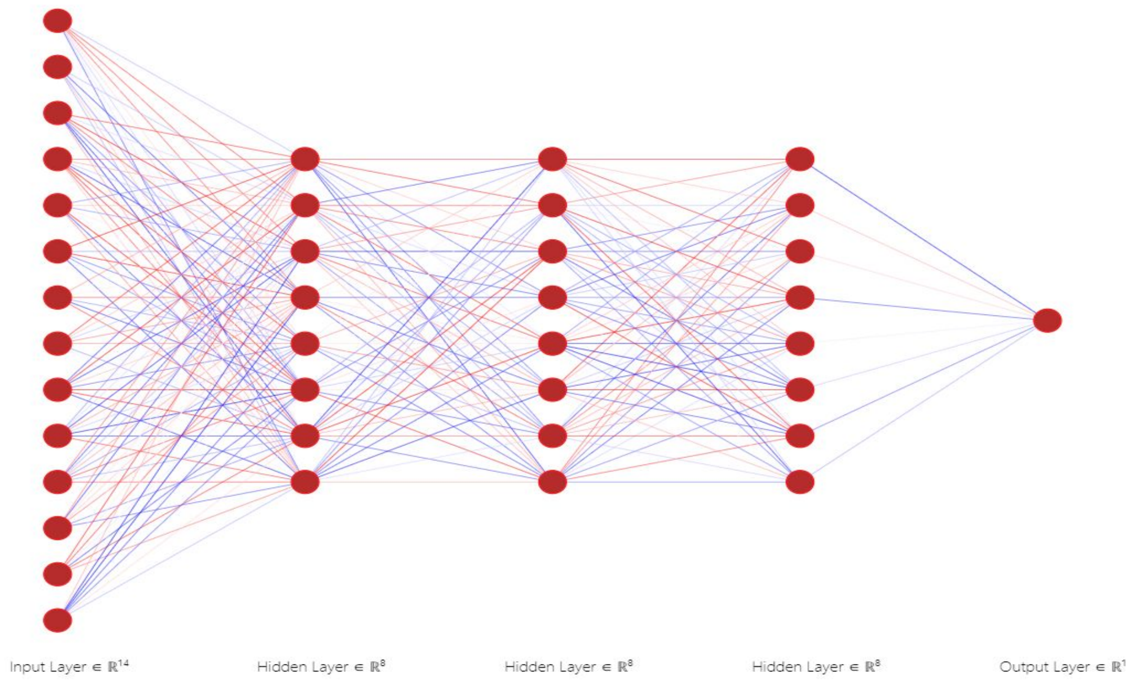


```
one_hot = pd.get_dummies(data['Shape'])
data = data.drop('Shape',axis = 1)
data = data.join(one_hot)
data = data.rename(columns={1.0: "Shape_1", 2.0: "Shape_2", 3.0: "Shape_3", 4.0: "Shape_4"})
```

Artificial Neural Networks(A.N.N)

- The model used was a Feed forward Neural Network, more specifically a Multilayer Perceptron.

- Its construction was carried out with Keras software, which works based on Tensorflow.



Input Layer $\in \mathbb{R}^{14}$
Hidden Layer $\in \mathbb{R}^{8}$
Hidden Layer $\in \mathbb{R}^{8}$
Hidden Layer $\in \mathbb{R}^{8}$
Output Layer $\in \mathbb{R}^{1}$

A.N.N Overview

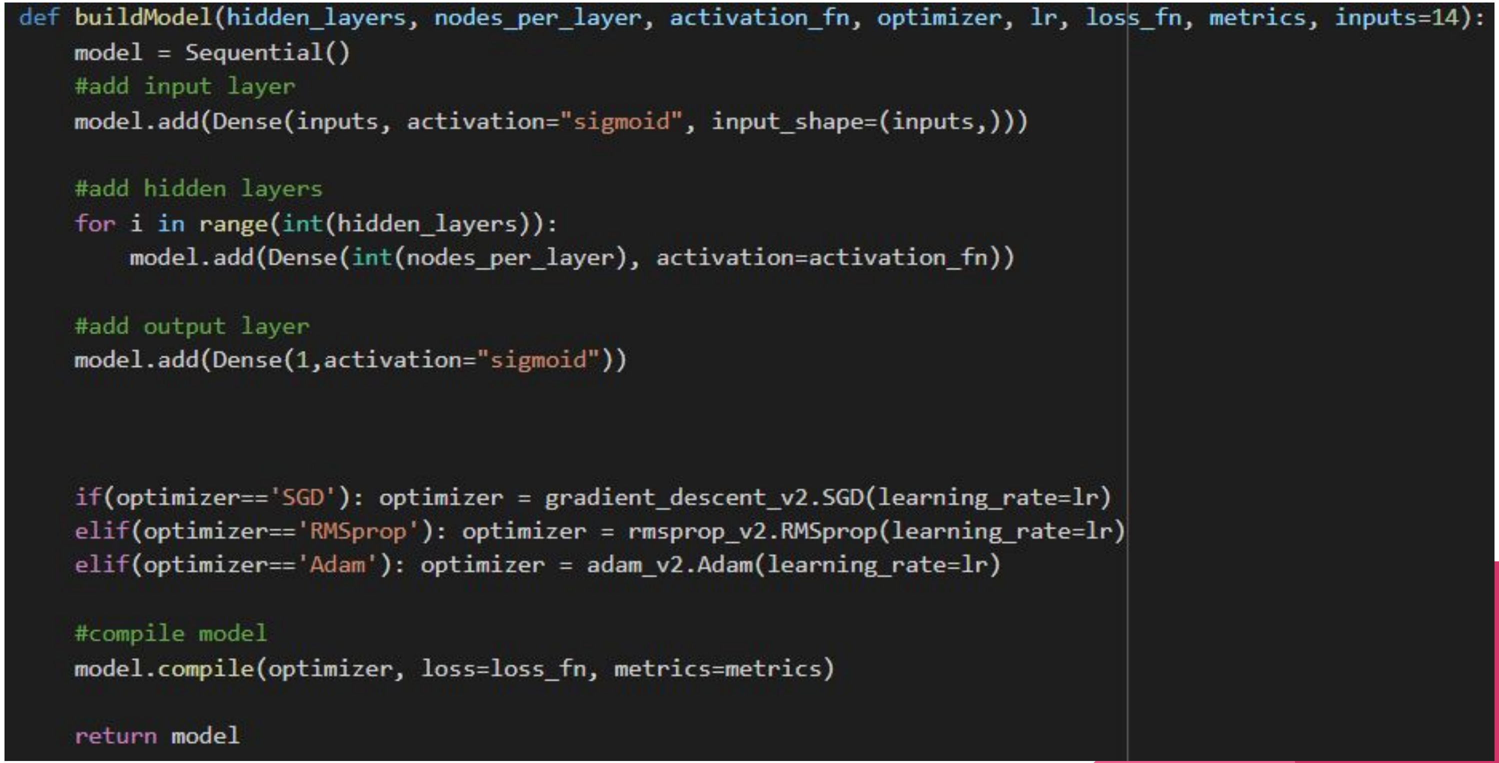
1. The network presents 14 input nodes that are, specifically: Age, Margin's 5 hypotheses, Shape's 4 hypotheses and Density's 4 hypotheses. . It presents a single output node, in this case the Severity target.
2. The input layer has the sigmoid activation function, as does the output layer.
3. In this way, we guarantee that the values are differentiated in a binary way in the first layer and that the values of the output layer are between 0 and 1.
4. All other parameters of the network architecture are selected. in an optimized way by a genetic algorithm, presented in the next section.
5. In order to avoid overfitting with the dataset in which the training is taking place, Cross Validation techniques were used.

Genetic Algorithm

- A genetic algorithm is a research heuristic inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the natural selection process, where the fittest chromosomes are selected for reproduction, in order to produce offspring of the next generation.
- One of the main processes in this algorithm is natural selection. This begins with the selection of the fittest chromosomes from a population, which produce ”children” that inherit the characteristics of the ”parents” are added to the next generation.
- If the ”parents” do better, the ”children” will be better than the ”parents” and will have even better results. This process is repeated until, in the end, a generation with the fittest chromosomes is found.
- This notion can be applied to a research problem,Objectively, we consider a set of solutions to the problem and select the best one.

Steps of Evolution (G.A Algorithm)

First we created the initial architecture of the neural network:



```python
def buildModel(hidden_layers, nodes_per_layer, activation_fn, optimizer, lr, loss_fn, metrics, inputs=14):
    model = Sequential()
    #add input layer
    model.add(Dense(inputs, activation="sigmoid", input_shape=(inputs,)))

    #add hidden layers
    for i in range(int(hidden_layers)):
        model.add(Dense(int(nodes_per_layer), activation=activation_fn))

    #add output layer
    model.add(Dense(1,activation="sigmoid"))



    if(optimizer=='SGD'): optimizer = gradient_descent_v2.SGD(learning_rate=lr)
    elif(optimizer=='RMSprop'): optimizer = rmsprop_v2.RMSprop(learning_rate=lr)
    elif(optimizer=='Adam'): optimizer = adam_v2.Adam(learning_rate=lr)

    #compile model
    model.compile(optimizer, loss=loss_fn, metrics=metrics)

    return model
```

Constitution of chromosomes

- The genetic algorithm process starts with a set of chromosomes called population, in which each chromosome is a solution to the problem in question. Each of the chromosomes is characterized by a set of variable parameters called genes. The genes are compacted to form a chromosome (solution).
- We have taken 6 genes to form a chromosome.
- The possible parameters are the following:
  - Number of intermediate layers, with a possible value between 1 and 16.
  - Number of nodes per middle layer, with a value from among the following: 1, 2, 4, 8, 16, 32, 64, 128.
  - Activation function, among the following: relu, selu, sigmoid, tanh, linear, softmax.
  - Learning rate, with a value from the following: 0.001, 0.01, 0.1
  - Optimization function, among the following: SGD, RMSprop, Adam
  - Loss function, from among the following: binary cross-entropy, hinge, squared hinge

Chromosome evaluation

After choosing, in a first random phase, the parameters included in the chromosomes, a neural network is trained with each of these sets of parameters.

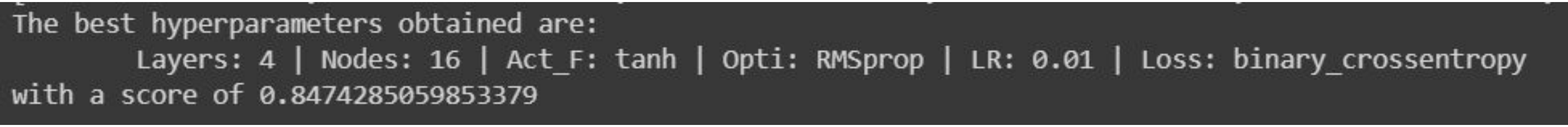
In the network, each chromosome is validated with K-Fold Cross Validation, applied over the training data set, previously separated through the test train split function.

The score of a particular chromosome is determined by the following expression:

score = f-beta_score with beta equal to 2

Explanation of the Score:

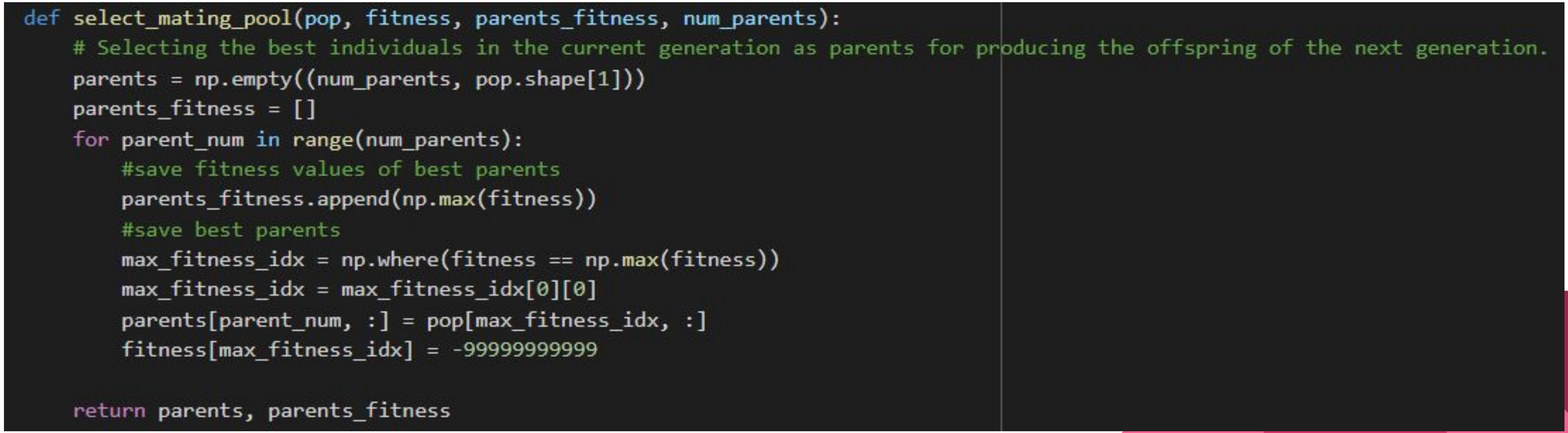
Given the context of the problem, it is important to try to minimize the number of false negatives(cases that the developed classifier model would classify as benign cancers, but would actually be malignant tumors). Thus we have given beta equal to 2



```
The best hyperparameters obtained are:
        Layers: 4 | Nodes: 16 | Act_F: tanh | Opti: RMSprop | LR: 0.01 | Loss: binary_crossentropy
with a score of 0.8474285059853379
```

Selection phase

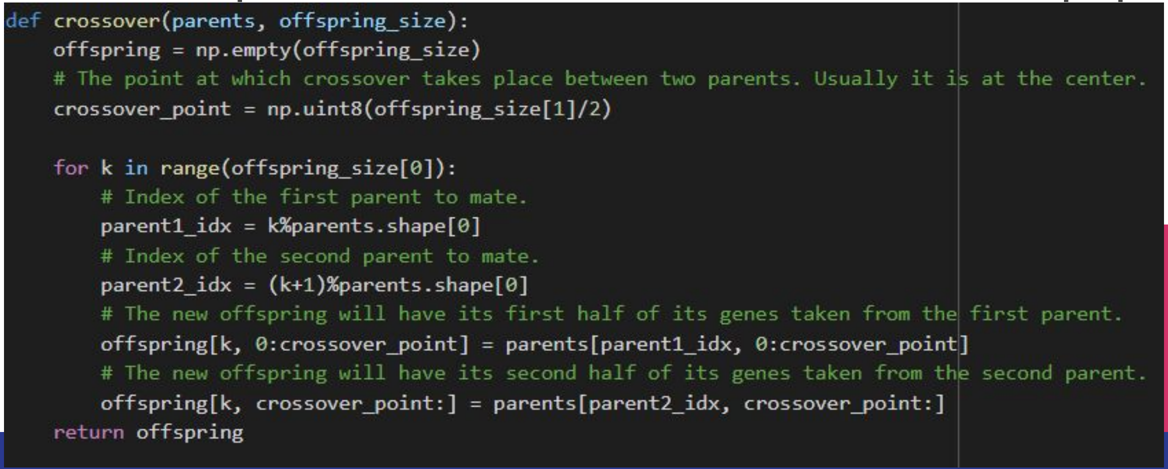
The idea of the selection phase is to select the chromosomes with the best results and allow them to pass their genes onto the next generation. In this way, the five chromosomes with the best score are selected to proceed to the next generation.



```
def select_mating_pool(pop, fitness, parents_fitness, num_parents):
    # Selecting the best individuals in the current generation as parents for producing the offspring of the next generation.
    parents = np.empty((num_parents, pop.shape[1]))
    parents_fitness = []
    for parent_num in range(num_parents):
        #save fitness values of best parents
        parents_fitness.append(np.max(fitness))
        #save best parents
        max_fitness_idx = np.where(fitness == np.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = -99999999999

    return parents, parents_fitness
```

Crossover

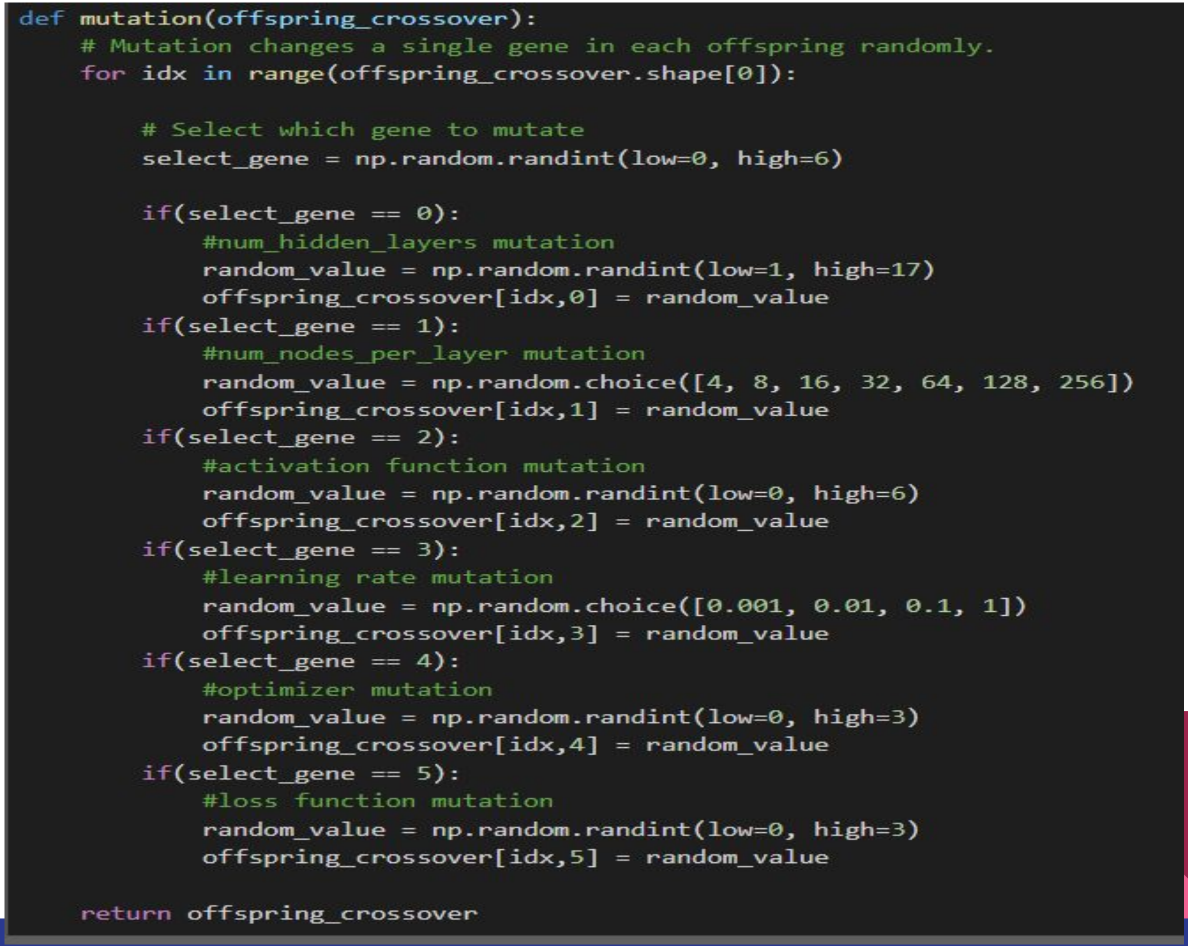
This is the most important phase of a genetic algorithm. For each set of "parent" chromosomes, a crossover point is randomly chosen from among the genes(since we had only 6 genes so we always selected the mid-point to reduce time to train) . The “offspring” are created by exchanging the genes of the parents with each other until the crossover point is reached and then added to the population.



```
def crossover(parents, offspring_size):
    offspring = np.empty(offspring_size)
    # The point at which crossover takes place between two parents. Usually it is at the center.
    crossover_point = np.uint8(offspring_size[1]/2)

    for k in range(offspring_size[0]):
        # Index of the first parent to mate.
        parent1_idx = k%parents.shape[0]
        # Index of the second parent to mate.
        parent2_idx = (k+1)%parents.shape[0]
        # The new offspring will have its first half of its genes taken from the first parent.
        offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
        # The new offspring will have its second half of its genes taken from the second parent.
        offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring
```

Mutation

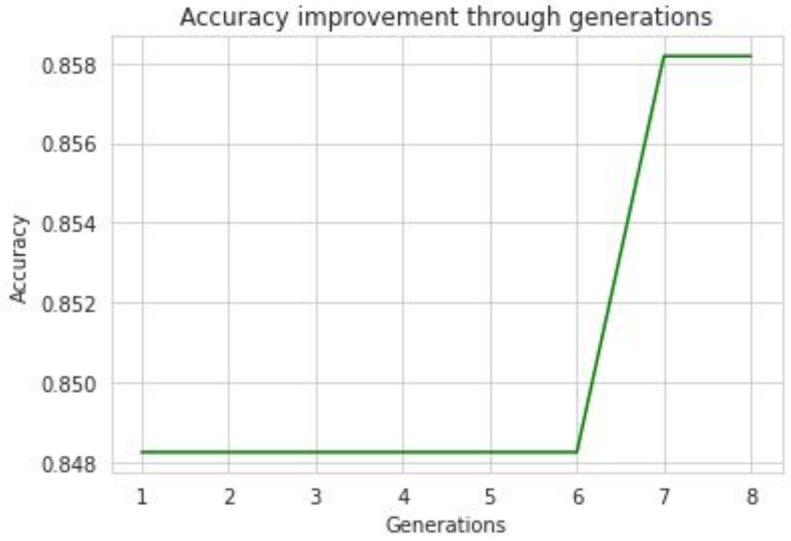
In the formed “daughter” chromosomes, some of the genes undergo a random mutation. This is done in order to maintain diversity within the population and prevent premature convergence.



```python
def mutation(offspring_crossover):
    # Mutation changes a single gene in each offspring randomly.
    for idx in range(offspring_crossover.shape[0]):

        # Select which gene to mutate
        select_gene = np.random.randint(low=0, high=6)

        if(select_gene == 0):
            #num_hidden_layers mutation
            random_value = np.random.randint(low=1, high=17)
            offspring_crossover[idx,0] = random_value
        if(select_gene == 1):
            #num_nodes_per_layer mutation
            random_value = np.random.choice([4, 8, 16, 32, 64, 128, 256])
            offspring_crossover[idx,1] = random_value
        if(select_gene == 2):
            #activation function mutation
            random_value = np.random.randint(low=0, high=6)
            offspring_crossover[idx,2] = random_value
        if(select_gene == 3):
            #learning rate mutation
            random_value = np.random.choice([0.001, 0.01, 0.1, 1])
            offspring_crossover[idx,3] = random_value
        if(select_gene == 4):
            #optimizer mutation
            random_value = np.random.randint(low=0, high=3)
            offspring_crossover[idx,4] = random_value
        if(select_gene == 5):
            #loss function mutation
            random_value = np.random.randint(low=0, high=3)
            offspring_crossover[idx,5] = random_value

    return offspring_crossover
```

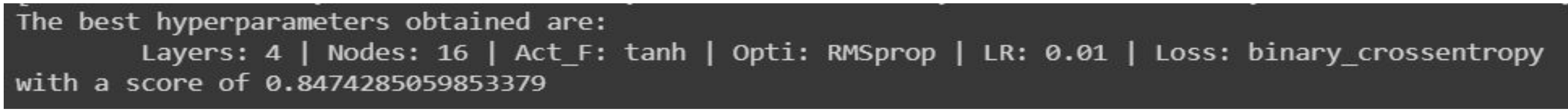
Performance of the Genetic Algorithm

In order to visualize the evolution of the score over the generations, the following graph is presented showing the improvement of this f-beta score with the execution of the algorithm.



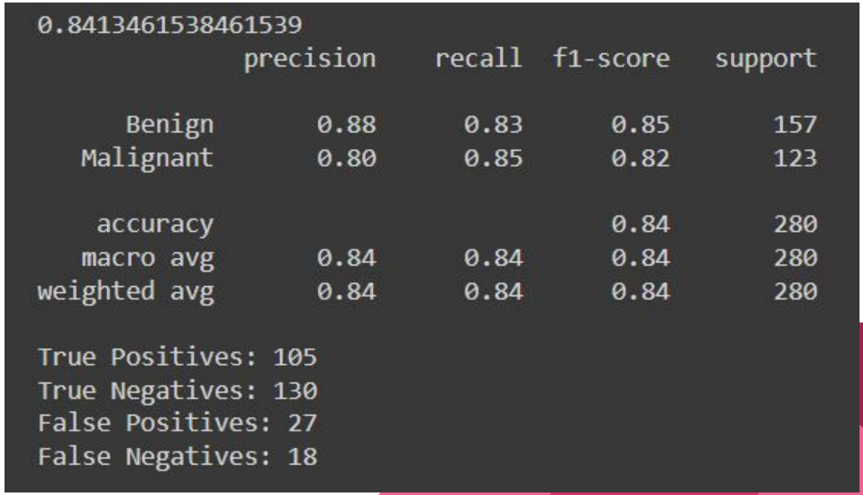
Results

Model Hyperparameters chosen were:



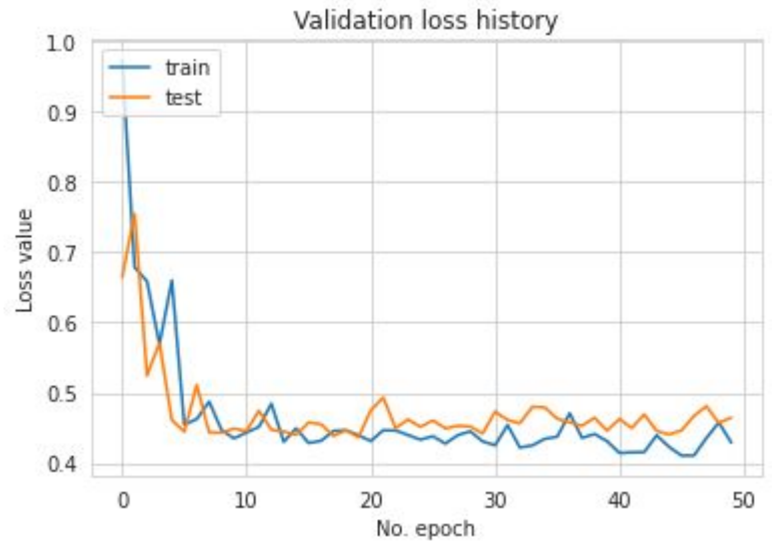
```
The best hyperparameters obtained are:
        Layers: 4 | Nodes: 16 | Act_F: tanh | Opti: RMSprop | LR: 0.01 | Loss: binary_crossentropy
with a score of 0.8474285059853379
```

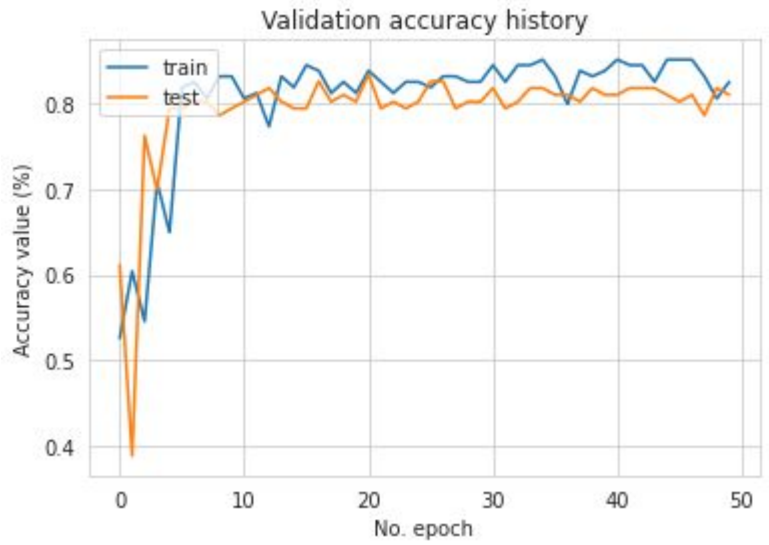
We finally trained the Model with final Parameters for 1500 epochs to find its absolute performance.



```
0.8413461538461539
              precision    recall  f1-score   support

      Benign       0.88      0.83      0.85       157
   Malignant       0.80      0.85      0.82       123

    accuracy                           0.84       280
   macro avg       0.84      0.84      0.84       280
weighted avg       0.84      0.84      0.84       280

True Positives: 105
True Negatives: 130
False Positives: 27
False Negatives: 18
```

The model showed 18 false-negative which were lesser than 26 false negative when the score function taken was f1-score, while still maintaining accuracy of 81%.


Validation loss history
train
test
Loss value
No. epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50


Validation accuracy history
train
test
Accuracy value (%)
No. epoch
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50

Conclusion

- After carrying out this work, we were able, in fact, to understand the results we obtained, taking into account the decisions taken throughout the project.
- One Hot Encoding and Standardization represent very important processes with regard to the efficiency of Neural Network algorithms and, as this model is constituted by an ANN, we also made the decision to carry them out.
- Regarding the neural network optimization process, we apply a genetic algorithm whose mutation between generations is based on a custom metric, developed with special attention to the context of the problem, with the objective of trying to minimize the number of false negatives.

Discussion

- Analyzing the results obtained, we can say that they are proportional to the effort involved in carrying out this work
- Certainly the application of the One Hot Encoding process would have been crucial to reach the values presented
- The main difficulties experienced involved learning all the parameters that can be used to configure the neural network, and a previous study was carried out, with some depth, which allowed us to reduce the scope of the solution search space.

References

UCI Machine Learning Repository: Mammographic Mass Data Set

A Gentle Introduction to the Fbeta-Measure for Machine Learning (machinelearningmastery.com)

Tackling Missing Value in Dataset - Analytics Vidhya

(PDF) Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms (researchgate.net)