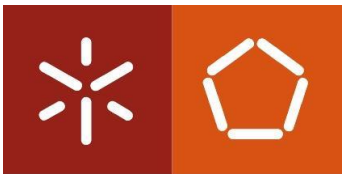

Universidade do Minho

Departamento de Informática

TP1 - GA para Otimizar Redes Neuronais

Bruno Martins (a80410), Catarina Machado (a81047),
Filipe Monteiro (a80229), Jéssica Lemos (a82061)

26 de Abril de 2020

Conteúdo

Resumo

Neste trabalho, utilizando o conhecimento adquirido em GA's e ANN's - *Genetic Algorithms* e *Artificial Neural Networks* - é nos proposto a criação de uma rede neuronal capaz de classificar um tumor a partir de um pedaço de massa mamária em benigno ou maligno, consoante as características deste. Para decisão dos parâmetros que irão constituir a rede neuronal, usamos algoritmos genéticos, de forma a tentar encontrar a melhor combinação de parâmetros possível.

1 Introdução

O mundo da medicina está em constante evolução. Uma dessas evoluções é a utilização de modelos de *machine learning* para o auxílio dos médicos na tomada de decisão acerca do diagnóstico feito a cada pessoa. O presente relatŕio foca-se na descrição das decisões tomadas para o auxílio da classificação de massas mamárias (benigna ou maligna) utilizando redes neuronais otimizadas recorrendo a algoritmos genéticos.

Numa primeira fase do projeto foi realizada uma análise dos dados de forma a que estes pudessem ser devidamente preparados para maximizar as chances de obter um modelo com uma eficácia alta. De seguida são descritos os passos realizados de forma a ter um algoritmo genético. Na quarta secção é feita uma breve descrição de como a rede neuronal funciona de forma a que seja possível integrar os resultados do algoritmo genético. Na secção seguinte é feita uma análise e otimização dos resultados obtidos seguida de uma breve conclusão em que é feita uma reflexão em relação a todo o o projeto.

2 Análise dos Dados

Este *dataset* é referente a um conjuntos de análises de mamografias, com vários campos, para detetar se o cancro presente é de origem benigna(0) ou maligna(1). Como tal, o nosso objetivo será tentar prever que tipo de cancro será com base num conjunto de parâmetros.

A primeira coisa que feita foi, ao importar os dados, retirar a coluna *BI-RADS* tal como era aconselhado, pois este servia como um medidor de confiança da severidade do problema (maior valor indicava que estava mais certo de que era determinado diagnóstico). Após isto, começamos por analisar várias componentes dos dados: existência de *missing values*, análise de correlação e dispersão destes.

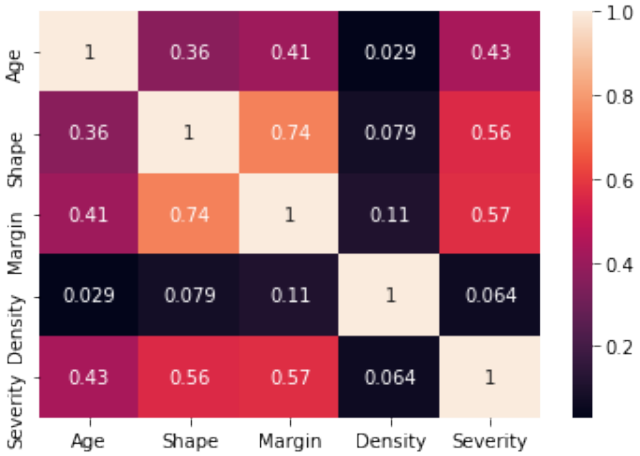

Figura 1: Correlação dos dados, antes de qualquer alteração.

Podemos verificar que o parâmetro *Density* possui um muito baixo valor de correlação para com a nossa variável de decisão *Severity*, mas dado que temos muitas poucas variáveis, iremos mantê-la, com a possibilidade de no fim, nos melhores modelos, experimentar treinar sem esta para verificar se existem melhorias. Ao carregar os dados verificámos também que existia uma quantidade considerada de *missing values* que precisavam ser tratados (cerca de 130 registos), sendo que os campos com valores em falta vão variando. Estes parecem estar em falta aleatóriamente e, como tal, decidimos eliminar estes registos e analisar se o *dataset* ficaria desbalanceado ou pior.

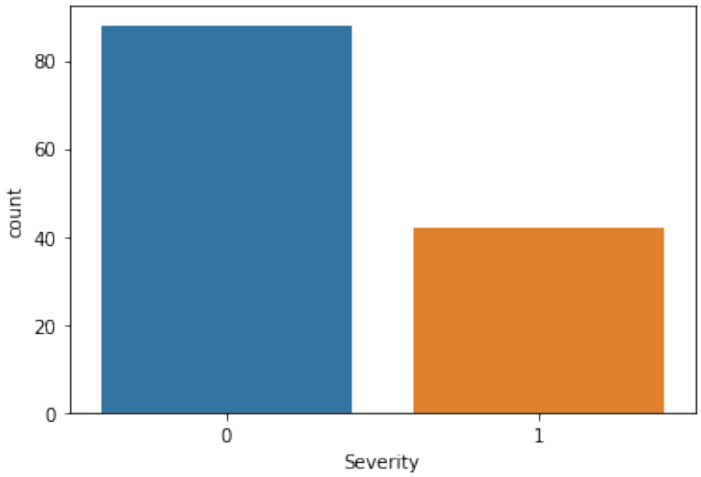

Figura 2: Proporção de casos positivos e negativos dos registos com falta de valores. A proporção vai de encontro à proporção do próprio *dataset*, sendo que não irá alterar para pior, mas para melhor.

	Age	Shape	Margin	Density	Severity
count	955.000000	929.000000	912.000000	884.000000	960.000000
mean	55.475393	2.721206	2.793860	2.910633	0.462500
std	14.482917	1.243428	1.565702	0.380647	0.498852
min	18.000000	1.000000	1.000000	1.000000	0.000000
25%	45.000000	2.000000	1.000000	3.000000	0.000000
50%	57.000000	3.000000	3.000000	3.000000	0.000000
75%	66.000000	4.000000	4.000000	3.000000	1.000000
max	96.000000	4.000000	5.000000	4.000000	1.000000

Figura 3: Distribuição dos dados antes de qualquer alteração.

	Age	Shape	Margin	Density	Severity
count	830.000000	830.000000	830.000000	830.000000	830.000000
mean	55.763855	2.783133	2.812048	2.915663	0.484337
std	14.667185	1.243057	1.565875	0.350936	0.500056
min	18.000000	1.000000	1.000000	1.000000	0.000000
25%	46.000000	2.000000	1.000000	3.000000	0.000000
50%	57.000000	3.000000	3.000000	3.000000	0.000000
75%	66.000000	4.000000	4.000000	3.000000	1.000000
max	96.000000	4.000000	5.000000	4.000000	1.000000

Figura 4: Distribuição dos dados depois de retirar os *missing values*.

Em relação aos *missing values* estes não aparentam ter dados importantes para o *dataset*, sendo que a dispersão dos valores de cada variável não variam praticamente nada. De facto, até ajudou a fazer um balanceamento da variável de decisão.

3 Algoritmo Genético

O algoritmo genético será a base do projeto, sendo esta que será executada para decisão, construção e análise das várias redes neuronais. Foi estipulada uma população de 15 soluções, sendo que o algoritmo irá parar ao fim de 75 execuções.

3.1 Representação das Soluções

Uma solução terá de ter uma variedade de parâmetros (genes) relativos a possíveis parâmetros da rede neuronal a otimizar. Entre elas:

- Nº de camadas intermédias;
- Nº de Nodos das camadas;
- Batch-size;
- Learning Rate;
- Função de ativação das camadas intermédias;
- Função de otimização;
- Função de loss;

Para além destas, tem um ID e a geração a que corresponde, assim como, depois da execução da rede neuronal com os seus parâmetros, os respectivos valores de **accuracy, falsos positivos e falsos negativos**.

3.2 Função de Fitness

Para cálculo de *fitness* foi criada uma função que dependia dos valores de *accuracy*, percentagem de falsos positivos e falsos negativos. A decisão tomada foi de minimizar o número de falsos positivos, tendo em conta que o projecto é de natureza médica, era importante classificarmos mais vezes incorrectamente positivamente do que negativamente. Assim, atribuímos 70% do peso ao valor da *accuracy*, sendo depois descontada por 20 e 10% do valor de falsos positivos e falsos negativos, respectivamente:

$$F(x) = 0.7 * ACC - 0.2 * FP - 0.1 * FN$$

No melhor dos casos, com 100% *accuracy* e 0 falsos positivos e falsos negativos, iría ter *fitness* 70, mas analisando mais realisticamente, um resultado ideal seria à volta de 94% de *accuracy* e alguma percentagem residual para os falsos positivos/negativos, resultando numa *fitness* entre 64-66.

3.3 Função de Seleção

Para o processo de seleção dos indivíduos a criar descendência, foi escolhido um processo baseado numa "Roda da Sorte: **Stochastic universal sampling**. Cada indivíduo terá uma probabilidade de ficar na nova geração, mas esta probabilidade varia consoante o *fitness* de cada um. Este método foi escolhido porque permite a existência de possíveis soluções menos boas que contenham talvez alguns bons genes para passar a outro descendente e talvez criar uma melhor solução. No entando, fizemos também uma versão diferente de seleção, onde se selecionava apenas os n melhores, para ver a diferença causada no nosso contexto.

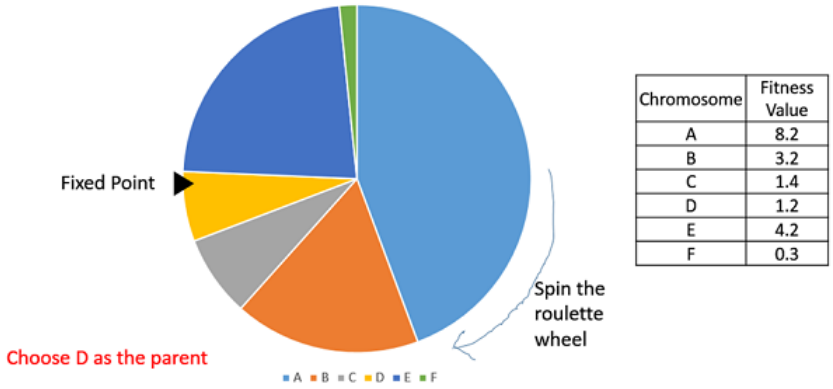

Figura 5: Pipeline do processo de otimização.

3.4 Função de Crossover

Após selecionada metade da geração a manter para gerar descendentes, o processo de *crossover* é então executado em cada par possível desta. Este *crossover* é executado usando **Uniform Crossover**, onde, para cada gene do descendente, é escolhido aleatoriamente de qual dos pais irá recebê-lo.

3.5 Função de Mutação

Em termos de mutação definiu-se que 20% de probabilidade de ocorrer era equilibrado, resultando em, por cada 10 soluções, 2 iriam sofrer mutação. Esta ocorre em apenas num dos genes que possui, tendo todas estas a mesma probabilidade de ocorrer.

4 Rede Neuronal

A rede neuronal foi construída através da utilização da biblioteca *keras* que fornece uma interface mais fácil e intuitiva para os utilizadores comparativamente a utilizar apenas *tensorflow*. Para a definição da rede neuronal foi criada uma função que tem como parâmetro um indivíduo da geração que contém todos os parâmetros necessários para construir a rede neuronal. Estes parâmetros são:

- Epochs: 200,250,300,350,400,450,500
- Activation: 'elu','relu','tanh'
- Optimizers: 'adam','sgd','adagrad','adadelta','rmsprop'
- Losses: 'binary crossentropy'
- Learning Rate: 0.1,0.01,0.001,0.0001
- Batch size: 32,64,128
- Neurons: 8,16,32,64
- N_Layers: 1,2,3,4,5

Desta forma a rede neuronal vai variar tendo em conta as características de cada individuo e as suas mutações tendo em conta os parâmetros acima descritos.

O *learning rate* apesar de estar como parâmetro de entrada na rede neuronal com um conjunto possível de valores, na verdade, foram testados manualmente cada um deles e escolhido como valor final constante apenas um deles para não existirem tantas variáveis na seleção de cada individuo.

5 Resultados

Em todas as execuções do processo de otimização da rede neuronal, foram usados os seguintes parâmetros:

- População de 15 individuos;
- 75 iterações
- *Threshold* para resposta ser positiva de 0.5%

Note-se que em termos da tomada de decisão de paragem do algoritmo colocaram-se algumas condições tal como tentar que a população estabilizasse numa configuração, mas isto provou ser difícil de conseguir. Desta forma, 75 rondas pareceram suficientes para este caso em específico. Outro aspecto importante é que não deixámos a rede a treinar durante muito tempo nem a executámos com *cross-validation*, porque apenas nos interessa saber o potencial de cada arquitetura, fazendo as comparações finais com mais atenção no fim da execução deste programa.

O nosso pipeline será então o seguinte:

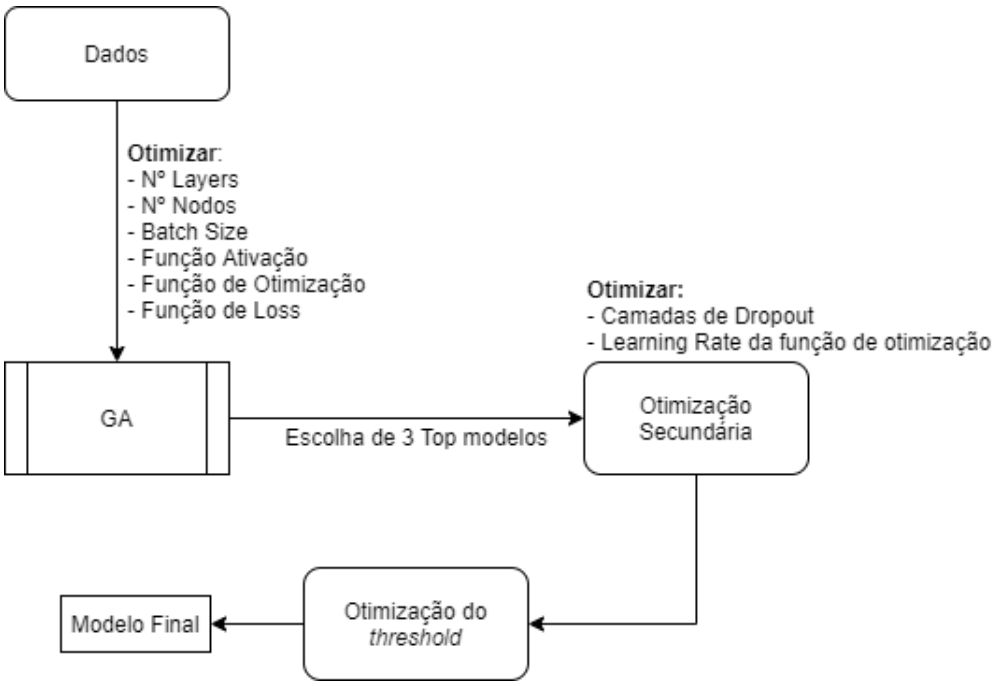

Figura 6: Pipeline do processo de otimização.

Correndo então o algoritmo genético, verificámos que ao fim de 20 iterações conseguimos durante várias iterações seguintes indivíduos com muito bons resultados, acabando por piorar mais tarde mas parecendo voltar a aumentar se aumentássemos o número de iterações. Apesar de o algoritmo não ter convergido numa única solução (o que faz sentido dado que muitas arquiteturas irão dar valores muito semelhantes), através da análise dos gráficos apresentados nas Figura 7 e 8 verificamos que existem indivíduos muito bons, podendo ser essa a melhor solução.

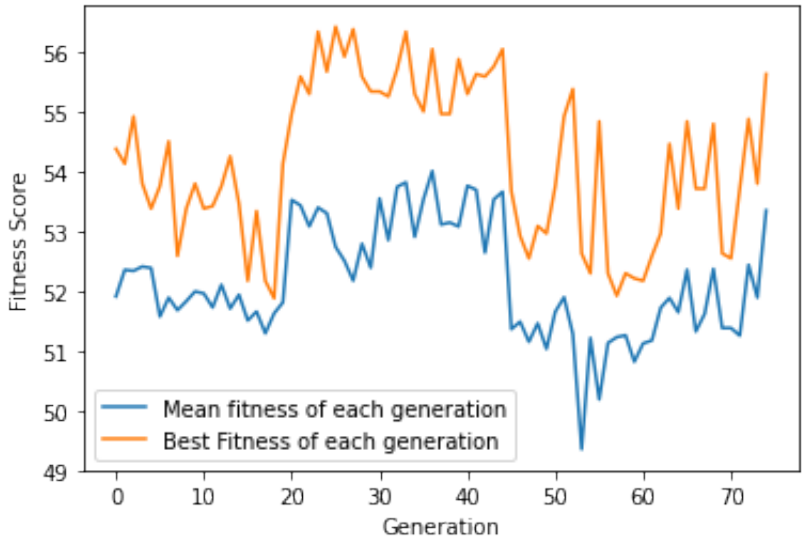

Figura 7: Progressão das gerações e dos *fitness* ao longo destas.

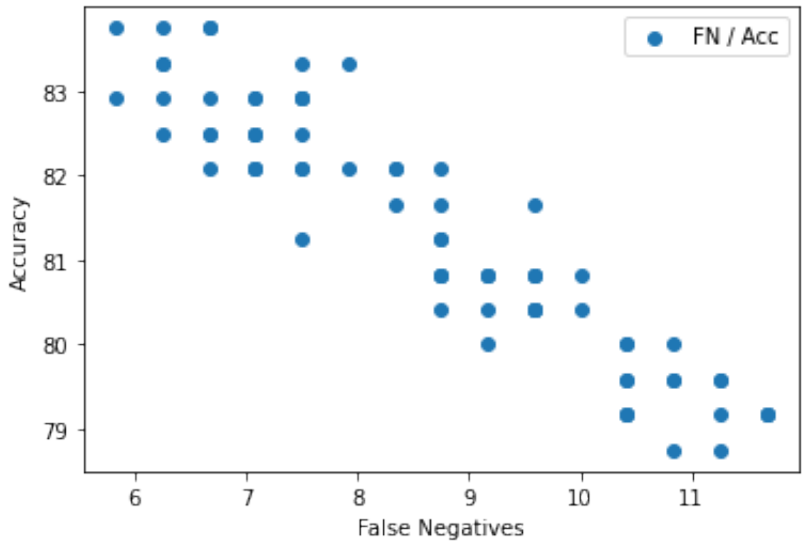

Figura 8: Melhores indivíduos de cada geração, em termos de número de falsos negativos e a sua *accuracy*.

Analisando o gráfico das melhores soluções de cada geração apresentado na Figura 8, verificámos que existem muitas acima de 82% *accuracy* e que, tendo em conta o número de falsos negativos, não parecem ser muito diferentes. As arquiteturas que iremos agora tentar otimizar manualmente serão apenas as 3 melhores do gráfico (pontos que constituem a diagonal mais à esquerda possível). Estes são:

```
Generation  25  ID  388
Fitness Score:  56.41666499773661
Layers:  4
```

```
Nodes:  64
Batch Size:  32
Activation_Function:  tanh
Optimizer:  rmsprop
Losse Function:  binary_crossentropy
FP:  10.416666666666668  | FN:  5.833333333333333  | ACC:  83.74999761581421
--------------------------------------------------
--------------------------------------------------
Generation  27  ID  419
Fitness Score:  56.37499833106994
Layers:  4
Nodes:  64
Batch Size:  64
Activation_Function:  tanh
Optimizer:  rmsprop
Losse Function:  binary_crossentropy
FP:  10.0  | FN:  6.25  | ACC:  83.74999761581421
--------------------------------------------------
--------------------------------------------------
Generation  43  ID  659
Fitness Score:  55.74999888737996
Layers:  4
Nodes:  64
Batch Size:  64
Activation_Function:  tanh
Optimizer:  adam
Losse Function:  binary_crossentropy
FP:  11.25  | FN:  5.833333333333333  | ACC:  82.91666507720947
--------------------------------------------------
--------------------------------------------------
```

Executando *cross validation* de $\mathbf{K = 5}$ para uma melhor noção da *accuracy*, obtemos os resultados na tabela 1. Passando agora para a segunda etapa da otimização, iremos então corre um algoritmo para testar algumas combinações aleatórias de *dropout* e *learning rate* da função de otimização para cada um dos indivíduos, assim como analisar mais de perto se o treino está a ser feito corretamente. Os resultados encontram-se também na Tabela 1. Em termos de arquitetura, ficaram com as seguintes mudanças:

- ID 388 - camadas *dropout* intermédias de 0.5 e *learning rate* de 0.01;
- ID 419 - camadas *dropout* intermédias de 0.2 e *learning rate* de 0.0001;
- ID 659 - camadas *dropout* intermédias de 0.2 e *learning rate* de 0.001;

	ID 388	ID 419	ID 659
ACC	78.1	79.2	80
FPS	12.7	13.3	12.9
FNS	9.2	7.6	7.1
Otimização dropout/learning rate			
ACC	80.5	80.2	81
FPS	12.4	11.8	11.3
FNS	7.1	8	7.7

Tabela 1: Resultado da *pipeline* completa.

Finalizando a *pipeline* com uma manipulação do *threshold* com o objetivo de minimizar os falsos negativos (são aqueles que mais importam neste caso médico, sendo que não queremos prever nenhum caso de negativo com a possibilidade de ser positivo), foram tentados 3 *thresholds* diferentes, visíveis na Tabela 2. O nosso objetivo é, como dito anteriormente, minimizar os falsos negativos, mas não existe nenhum *threshold* diferente de 0.5 que o faça sem prejudicar muito os outros, sendo que concluímos deixar este valor *standard*.

	ID 388	ID 419	ID 659
Otimização do Threshold = 0.4			
ACC	78.3	77.9	78.1
FPS	16	15	15.4
FNS	6.7	7.2	6
Otimização do Threshold = 0.45			
ACC	77.4	77.9	77.4
FPS	15.1	14	14.4
FNS	7.7	8	8.2
Otimização do Threshold = 0.5			
ACC	80.5	80.2	81
FPS	12.4	11.8	11.3
FNS	7.1	8	7.7
Otimização do Threshold = 0.6			
ACC	79.8	80.4	80.1
FPS	11.5	10.2	9.9
FNS	10.1	10.5	10.9

Tabela 2: Resultados para os diferentes *thresholds*.

De forma a testar outro tipo de algoritmo de seleção na *GA* para analisar se convergia ou descobria melhores soluções, decidimos então correr apenas 30 iterações onde em cada nova geração se escolhia para cruazmento os top *N*. Sem surpresa, este encontrou de facto uma solução melhor que o anterior, mas isto poderá ter sido apenas devido a aleatoriedade. Neste caso pegamos apenas numa solução, pois foi a única melhor que as anteriores:

```
_________________________________________________
_________________________________________________
Generation 6 ID 91
Fitness Score: 56.62499805291493
Layers: 7
Nodes: 16
Batch Size: 32
Activation_Function: elu
Optimizer: adam
Losse Function: binary_crossentropy
FP: 8.75 | FN: 7.083333333333333 | ACC: 84.16666388511658
_________________________________________________
_________________________________________________
```

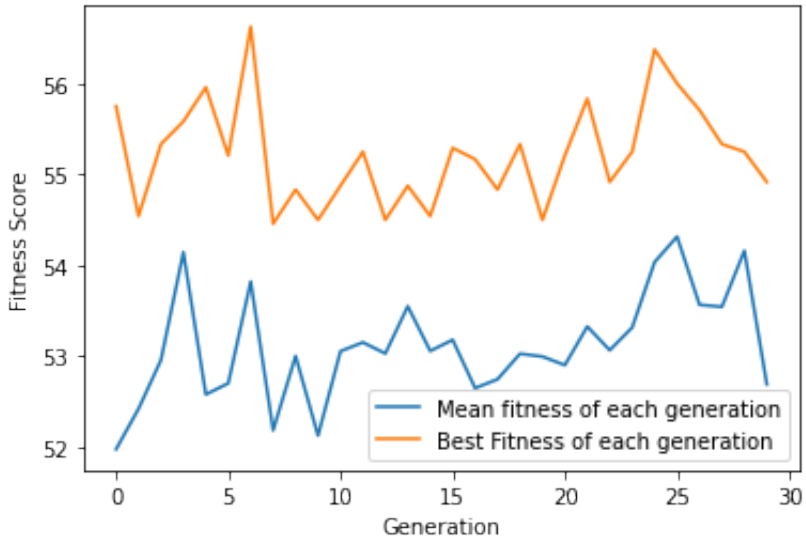

Figura 9: Progressão das gerações e dos *fitness* ao longo destas.

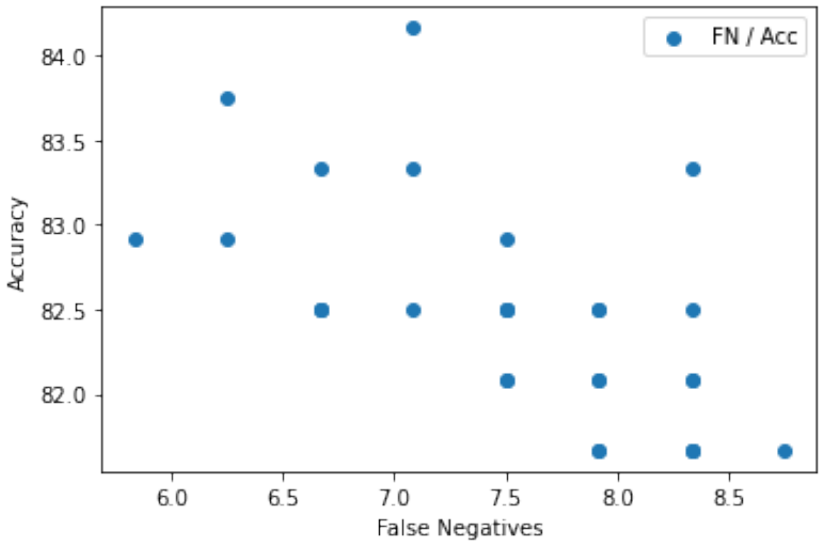

Figura 10: Melhores indivíduos de cada geração, em termos de número de falsos negativos e a sua *accuracy*.

Com isto repetimos o mesmo processo, obtendo como *dropout* 0.2 e *learning rate* 0.0001, conseguindo uma *accuracy* de 78.4%, 13.8 e 7.8% de falsos positivos e falsos negativos, respetivamente. Em termos de otimização do *threshold* deu resultados semelhantes às outra soluções, ligeiramente inferiores. Como tal, achamos que a arquitetura final deveria ser a do *ID 659*:

- 4 camadas com camadas intermédias de *dropout* de 0.2%;
- 64 nodos por camada;
- 64 de *batch*;
- *tahn* como função de ativação;

- *adam* como função de otimização, com *learning rate* de 0.001;
- *binary_crossentropy* como função de *loss*.

6 Conclusão

Algorimos genéticos são algoritmos que possibilitam um processo automático de otimização de modelos de *machine learning*, com um baixo custo de implementação, tendo apenas como defeito a quantidade de tempo necessário para treinar, crescendo facilmente com os modelos em questão. Com este trabalho aprendemos que dada as tantas possibilidades de arquiteturas possíveis para redes neuronais, escolher manualmente uma é dispendioso e não praticado, sendo que nos focámos pelas arquiteturas *tradicionais*, mas usando uma ferramenta por trás como as *GA's* este processo torna-se simples e com pouco esforço da nossa parte. É importante referir que a solução encontrada é apenas uma de muitas boas soluções que a *GA* encontrou, sendo que é possível existirem muitas mais ainda nesta nuvem de soluções.

Por fim, neste problema específico de natureza médica, a solução encontrada foi de encontro ao nosso objetivo de minimizar a possibilidade de erro médico, tendo apenas aproximadamente 7.7% de um falso negativo e 11.3% de falso positivo, com uma boa *accuracy* na casa dos 81%. Para obter mais resultados podíamos, por exemplo, experimentar o uso de várias redes na previsão (*ensembling*), no entanto esta técnica caia um pouco fora do âmbito do projeto.