Redes Neuronais Convolucionais em Classificação de Aves

Computação Natural
4º Ano - 2º Semestre
Mestrado em Engenharia Informática

Constança Elias, PG42820

21 de agosto de 2021

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

Este trabalho foi desenvolvido no âmbito da unidade curricular de Computação Natural. Os objetivos do trabalho consistem em: analisar e preparar um *dataset* de espécies de aves previamente fornecido; criar um modelo de redes neuronais convolucionais capaz de classificar corretamente as espécies de aves do *dataset* e desenvolver um algoritmo genético para otimizar os hiperparâmetros do modelo de aprendizagem criado, de modo a chegar ao modelo final para avaliação dos resultados.

As redes neuronais convolucionais são o estado de arte no que toca aos algoritmos de *machine learning* utilizados na classificação de imagens e não só [1]. O poder do uso de *Transfer Learning* na análise e classificação de dados tem sido apresentado na literatura como um método muito vantajoso na classificação de imagens[2]. Esta análise irá ser feita mais à frente.

1.2 Estrutura do Relatório

Este relatório divide-se em sete capítulos.

O Capítulo 1 introduz o trabalho proposto e os objetivos que se pretendem alcançar no final.

O Capítulo 2 faz uma análise detalhada do problema e explica como foi feita a preparação do *dataset*.

No Capítulo 3 é explicada a primeira fase do desenvolvimento do modelo.

O Capítulo 4 corresponde ao desenvolvimento do algoritmo genético para otimizar os hiper-parâmetros que definem a rede neuronal.

O Capítulo 5 apresenta os resultados referentes à avaliação do modelo final escolhido.

No Capítulo 6 é feita a discussão dos resultados obtidos.

Por fim, no Capítulo 7 é feita uma breve síntese do trabalho realizado, apresentando as principais conclusões do mesmo.

Capítulo 2

Análise e Preparação do *Dataset*

O dataset fornecido contém 250 espécies de pássaros, encontrando-se dividido em 3 diretorias para treino, validação e teste, respetivamente. No total, existem 3522 imagens para treino, 1250 para teste (5 por espécie) e 1250 para validação (5 por espécie). Fazendo uma análise inicial ao número de imagens por classe (ver figura 2.1), podemos constatar que o dataset não se encontra balanceado, sendo que a média de imagens por classe é 140 e a classe com menor número de imagens contém 95 imagens enquanto que a maior contém 300 imagens. Podemos então concluir que o *dataset* se encontra desbalanceado.

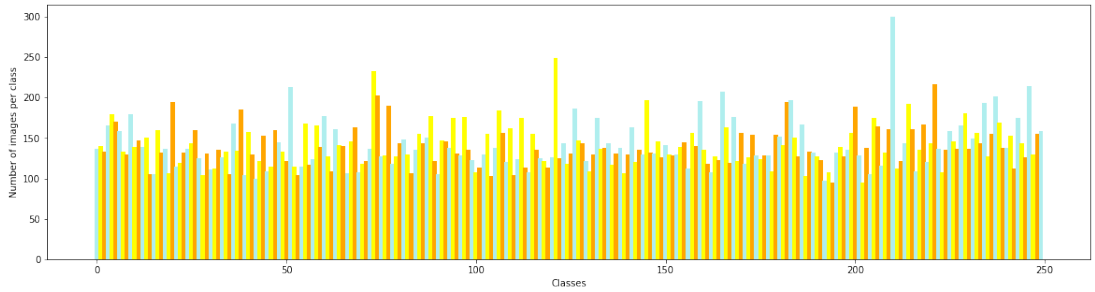

Figura 2.1: Análise do número de imagens por classe no *dataset* de treino

Analisando as imagens de algumas classes (figura 2.2), podemos ver que a maioria das aves se encontra centrada na imagem, havendo exceções.

Figura 2.2: Amostra do *dataset* de treino

Para fazer o *upload* das imagens para um *generator*, decidimos utilizar a função `flow_from_directory`, usando como argumentos o tamanho da imagem e modo de classificação de imagens (`sparce`), deixando o *batch size* como argumento a ser passado aquando da fase de treino do modelo desenvolvido.

2.1 *Image Augmentation*

Tendo em conta as características analisadas, ponderámos usar *Image Augmentation* para tentar colmatar o problema. No entanto, uma vez que o *dataset* de treino já possui bastantes imagens, optámos por não utilizar este método pelo menos numa primeira fase, tendo em conta o peso computacional acrescido que esta operação iria trazer e o facto de alguns estudos feitos recentemente apontarem que esta processo não revelou ser vantajoso nos resultados obtidos [3].

Capítulo 3

Primeira Fase da Resolução - CNN

Numa primeira abordagem do problema decidimos criar uma rede neuronal convolucional, tal como proposto, para avaliar os resultados. Começámos por utilizar uma arquitetura idêntica a uma rede definida em aula, para obter os primeiros resultados, avaliando as métricas de *accuracy* e *loss*.

Os resultados revelaram-se muito fracos (acurácia $< 1\%$, ver figuras 3.1 e 3.2), o que pareceu ser normal uma vez que foi a primeira definição do modelo e tendo em conta as características do *dataset* e do problema em questão.

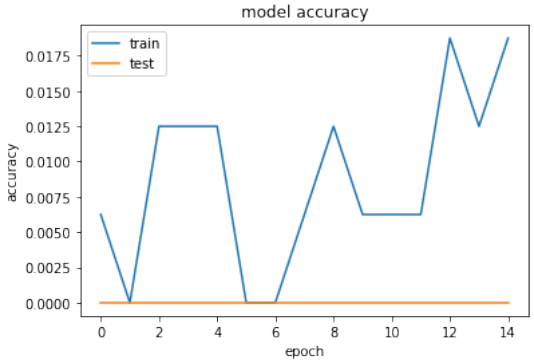

Figura 3.1: Valores de acurácia do primeiro modelo de CNN

Para função de *loss*, decidimos utilizar a função `sparce categorical entropy`, uma vez que, comparativamente com a *categorical entropy*, poupa memória e operações computacionais pois apenas utiliza um inteiro para classificar cada classe em vez de um vetor.

Com a discussão de resultados entre colegas e após treinar algumas variações deste rede, decidimos abandonar a ideia de definir uma rede neuroral de raíz. Após alguma pesquisa e com o incentivo do professor, decidimos aplicar *Transfer Learning* para tentar obter melhores resultados, tal como sugerido para *datasets* muito grandes [4]. Na secção seguinte iremos apresentar e explicar os modelos utilizados.

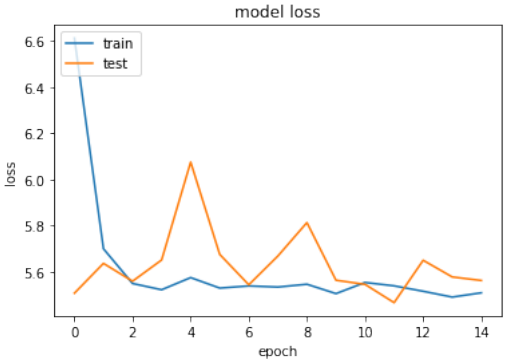

Figura 3.2: Valores de *loss* do primeiro modelo de CNN

3.1 *Transfer Learning*

Transfer Learning é o processo de utilizar um modelo pré-treinado num problema relacionado. Em *deep learning*, a aprendizagem por transferência é uma técnica em que um modelo de rede neural é primeiro treinado sobre um problema semelhante ao problema que está a ser resolvido. Uma ou mais camadas do modelo treinado são então utilizadas num novo modelo que será treinado sobre o problema de interesse. Uma das grandes vantagens da incorporação deste processo no nosso problema será a diminuição bastante significativa do tempo de treino da rede que iremos definir. Aquando do treino do nosso modelo, as primeiras camadas (de *feature extraction*) serão "congeladas", sendo apenas treinadas as camadas definidas por nós, as *fully connected layers*. Optámos por testar vários modelos pré-treinados já existentes. Tanto o `VGG16` como o `VGG19` e o `ResNet50` usam imagens de tamanho 224 x 224 enquanto que o `InceptionV3` usa imagens de 299 x 299 pelo que optámos por usar os três primeiros e assim evitar esforço computacional extra para adaptar as imagens para o tamanho requirido pelo `InceptionV3`.

3.1.1 ResNet101V2

Este modelo de *transfer learning* é apresentado em alguns estudos atuais como tendo apresentado melhores resultados [3], pelo que optámos por dar mais ênfase a este modelo. O problema proposto em [3] é muito semelhante ao problema de classificação que estamos a tentar resolver pelo que decidimos logo averiguar os resultados do modelo aplicado ao nosso conjunto de imagens. O primeiro treino deste modelo, com os mesmos parâmetros definidos em [3], com apenas 5 *epochs*, demorou cerca de 2 horas e 13 minutos a executar, tendo obtido 92% de acurácia no *dataset* de validação. As figuras 3.3 e 3.4 apresentam os resultados obtidos.

Tendo em conta os resultados obtidos para este modelo decidimos avançar para a definição do algoritmo genético utilizando este modelo pré-treinado como modelo a otimizar. Iremos este explicar este algoritmo detalhadamente na próxima secção.

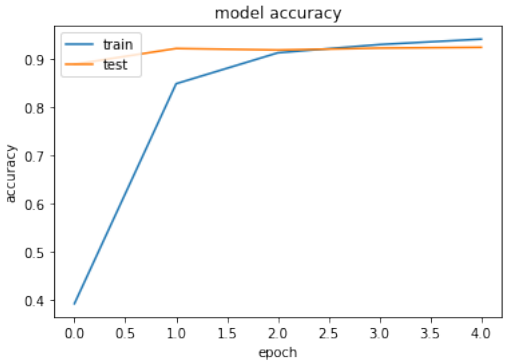

Figura 3.3: Valores de acurácia do primeiro treino do modelo ResNet101V2

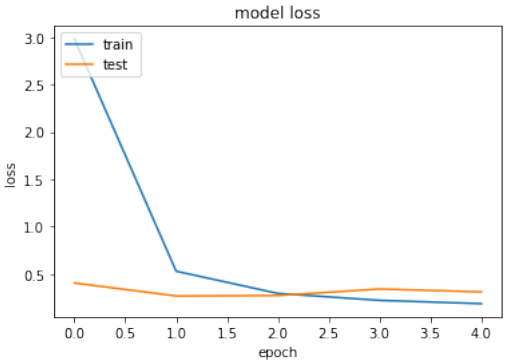

Figura 3.4: Valores de *loss* do primeiro treino do modelo ResNet101V2

Capítulo 4

Segunda Fase da Resolução - Algoritmo Genético

A segunda fase da resolução do problema consistiu em desenvolver um algoritmo genético para encontrar os melhores hiperparâmetros que iriam definir a rede escolhida previamente. Devido à limitação de recursos computacionais disponíveis e tratando-se esta etapa apenas de estudo e procura dos hiperparâmetros para definir a rede, cada modelo foi treinado apenas com 5 *epochs*, para tirar algumas conclusões iniciais.

4.1 Algoritmo Genético

O algoritmo genético (AG) foi desenvolvido com o objetivo de obter a maior diversidade de indivíduos, permitindo uma maior exploração das possíveis soluções e evitando uma convergência precipitada. As decisões tomadas relativamente à definição dos vários elementos que compõem o algoritmo foram feitas tendo em conta estudos recentes sobre o tema (como por exemplo em [5], [6]). Foi pensado para correr com 10 gerações e uma população de cerca de 15 indivíduos. No entanto, devido às limitações computacionais, acabámos por apenas utilizar 5 gerações e 10 indivíduos.

O algoritmo genético definido contém uma arquitetura simples (ver figura 4.1). Na geração inicial são inicializados 10 indivíduos e, para cada um, é treinado o modelo com os parâmetros que o define, com 5 *epochs*. De seguida, são selecionados os pais que irão dar origem aos dois filhos que farão parte da geração seguinte. Ao introduzir os novos filhos, são eliminados os dois indivíduos com os piores valores de *fitness* obtidos. Para critério de paragem do algoritmo definimos um *threshold* de 0.5. Caso o valor de *fitness* seja igual ou superior a 0.5, então o algoritmo termina.

De seguida, são explicadas as várias funções que compõem o AG.

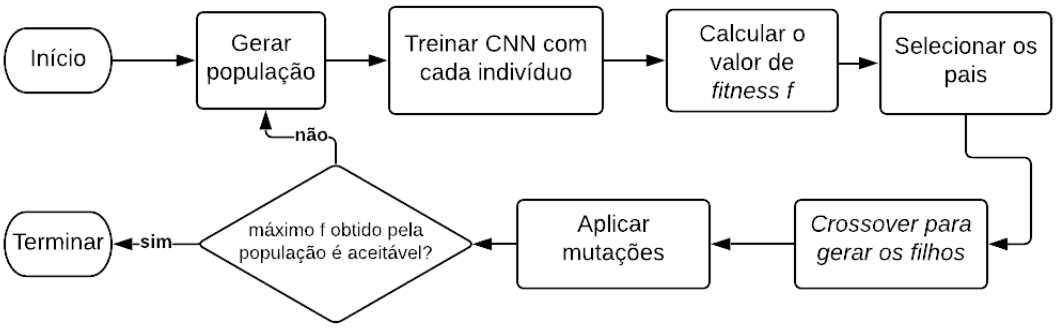

Figura 4.1: Arquitetura do Algoritmo Genético

4.1.1 Cromossoma

É constituído pelos parâmetros que iremos selecionar para definir a arquitetura do modelo escolhido, no sentido de otimizar a rede. Estes parâmetros são:

- função de ativação:
- função de otimização:
- tamanho do *batch*:
- *learning rate*:

Uma vez que estamos a utilizar um modelo pré-treinado, em que herdamos as camadas convolucionais, pensámos que não faria sentido avaliar o número de camadas. Para além disso, ao contrário do que foi pensado inicialmente, o número de *epochs* também acabou por não ser otimizado no algoritmo, pois recorremos à *callback* de *Early Stopping*, que se encarrega de parar o modelo caso o valor de *loss* comece a diminuir e o modelo deixe de aprender.

4.1.2 Função de Seleção

Após alguma pesquisa e tendo em conta o que foi feito nas aulas, percebemos que o método da *Roulette Wheel Selection* seria um bom método a ser utilizado porque permite obter uma maior diversidade de indivíduos e, consequentemente, uma melhor exploração das possíveis soluções. Em 2015, foi feito um estudo ([7]) que avalia as várias funções de seleção que podem ser aplicadas para otimizar os algoritmos genéticos. O *elite selection method*, que seleciona um número pré-determinado de cromossomas com os valores mais altos de *fitness* e que assegura que esse valor não se perde entre as gerações, revelou ser o melhor em termos de obtenção de acurária. A desvantagem deste método é que há pouca diversidade em termos de genes, porque há uma convergência maior entre as gerações. Por estas razões e pelos objetivos que definimos em cima, achámos mais adequado aplicar o outro método de seleção.

O método da *Roulette Wheel* cria uma distribuição de probabilidade ao selecionar os cromossomas que ficam na população. Os cromossomas com valor de *fitness* mais elevado são ponderados para que sejam selecionados mais vezes [8]. A figura 4.2 exemplifica o funcionamento deste método. Estabelece-se então um compromisso entre

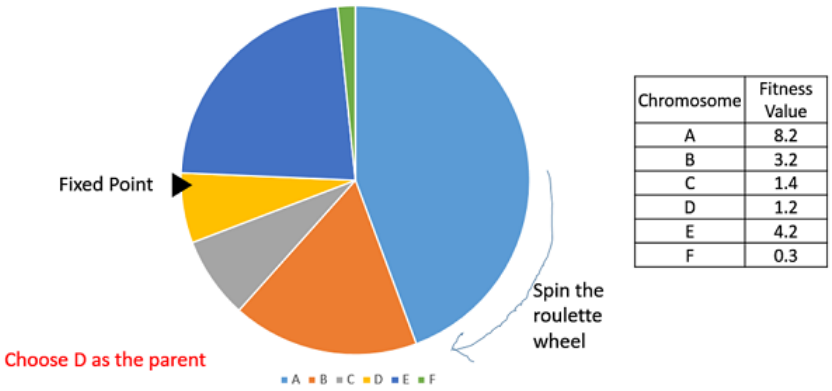

Figura 4.2: Funcionamento do *Roulette Wheel Selection method*

exploração e aproveitamento. A função desenvolvida, que implementa este algoritmo de seleção, foi baseada no código desenvolvido em [9].

4.1.3 Função de *Fitness*

A função de *fitness* que definimos consiste numa fórmula que calcula a diferença entre o valor da acurácia obtido e o valor de *loss*. Quanto menor o valor de *loss*, melhor é o modelo. Este valor, que não está em percentagem, diz-nos quão bom o modelo está a ir tanto no treino como no de validação, isto é, consiste num sumário dos erros feitos para cada imagem nos dois conjuntos de imagens. É, portanto, necessário ter esta métrica em conta.

4.1.4 Função de *Crossover*

A função de *crossover* foi definida utilizando o operador de *Uniform Crossover*, em cada cada gene da descendência é tratado separadamente e é escolhido aleatoriamente entre os genes dos pais. De entre os operadores possíveis, este foi o escolhido para poder criar uma maior diversidade.

4.1.5 Função de Mutação

Esta função consiste apenas em alterar um gene do cromossoma aleatoriamente, de acordo com uma certa probabilidade de ocorrência, no sentido de tornar o processo mais natural. Esta função também foi definida desta forma com o objetivo de permitir obter maior diversidade de soluções, não tendo em conta parâmetros ou critérios específicos.

4.1.6 Execução do AG

Devido a limitações computacionais, nomeadamente a utilização do GPU do *Google Colab*, não foi possível executar o algoritmo até ao fim, pelo que foi selecionado o melhor indivíduo de entre aqueles que foi possível gerar. A solução ótima encontrada será analisada na próxima secção.

Capítulo 5

Terceira fase da Resolução - Modelo Final

Na terceira e última fase, apurámos o modelo final selecionado, deixando-o convergir até ao fim para obter os resultados finais. Idealmente, gostaríamos de ter testado com um número elevado de *epochs* (a rondar as 100). No entanto, devido ao esgotamento dos recursos disponibilizados pelo *Google Colab*, não nos foi possível fazê-lo. Acabámos então por testar apenas com 10 *epochs*. O modelo escolhido foi o que obteve melhor valor de *fitness* na geração do AG que conseguimos executar (não foi possível executar o AG até ao fim e consequentemente obter, na prática, a melhor solução).

O indivíduo selecionado para a obtenção dos resultados finais, nesta terceira fase, possui as seguintes características:

- *Learning Rate* - 0.1
- *batch size* - 16
- função de ativação - *tanh*
- otimizador - *adadelta*

Para deixar convergir o modelo, fizemos um treino com 10 *epochs*, como foi referido. Os resultados obtidos em termos de *accuracy* e *loss* encontram-se nas figuras 5.1 e 5.2, respetivamente.

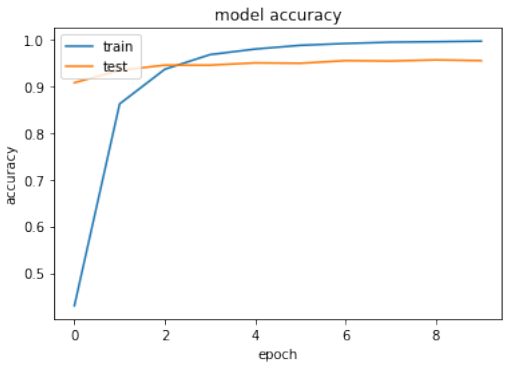

Figura 5.1: Valores de acurácia do modelo final

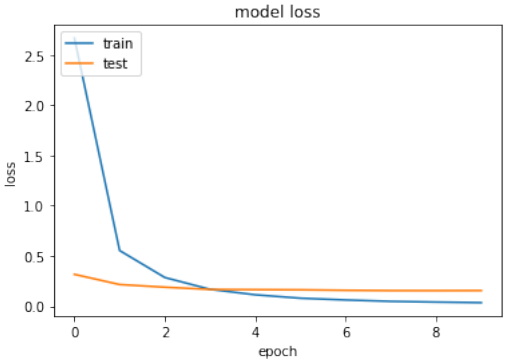

Figura 5.2: Valores de *loss* do modelo final

A avaliação do modelo final no *dataset* de teste obteve bons resultados: 97% de acurácia e 0.12 de *loss*. O facto de estes resultados não terem sido melhores poderá dever-se ao facto de que cerca de 4/5 do *dataset* de treino conter imagens de machos e 1/5 conter imagens de fêmeas.

5.1 *Predictions*

No sentido de avaliar mais a fundo os resultados obtidos, decidimos fazer algumas previsões no *dataset* de teste, usando para isso a função `predictions`. Observámos, no entanto, que as previsões não foram muito boas. Construímos um *heatmap*, com base na matriz de confusão, para melhor visualização dos resultados obtidos pelas previsões e, de seguida, um gráfico para visualização do valor de acurácia obtido nas mesmas.

Existem várias razões que poderão estar a causar estes resultados, nomeadamente: a função de ativação escolhida, valor de *loss* e o número de falsos positivos obtidos. Esta última métrica deveria ter sido explorada para averiguar melhor tais resultados. No entanto, devido à limitação do tempo disponível, esta não chegou a ser testada.

Capítulo 6

Discussão dos Resultados

Os resultados obtidos ao longo da definição dos vários modelos foram condicionados por um fator significativo: limitações de recursos computacionais. A ferramenta utilizada para treino dos modelos foi o *Google Colab*, mas este permite uma alocação limitada do GPU por utilizador. Para colmatar, em parte, este problema, decidimos alternar a execução do código pelas três contas *Google* que temos. Houve ainda um lapso adicional que levou ao aumento significativo do tempo de execução. A definição das diretoria dos três *datasets* foi mal feita, tendo sido definido o caminho para a pasta google drive e não do disco do *Colab*, tornando assim o processo mais lento. Só perto da entrega do trabalho é que esse lapso foi detetado e corrigido pelo que não nos permitiu usufruir do ganho de *performance* que esta correção trouxe.

Os resultados obtidos são bons, no entanto existe uma causa que poderá ter impedido melhores resultados. O facto de o *dataset* não estar proporcional relativamente ao número de imagens de machos e fêmeas. Os machos típicos têm cores muito mais diversificadas, enquanto que as fêmeas de uma espécie são tipicamente menos coloridas. Consequentemente, as imagens de machos e fêmeas podem parecer totalmente diferentes e, portanto, dificultar a classificação por parte do modelo.

No que toca ao algoritmo genético, com acesso a uma máquina com maiores recursos computacionais, teríamos feito mais testes, com mais gerações, e teríamos definido uma população maior. Para além disso, estenderíamos o número de parâmetros a otimizar.

A nível de maior exploração de soluções, para colmatar o problema da falta de recursos, poderiamos ter optado por diminuir o número de espécies (classes) a avaliar no modelo, com o objetivo de tentar conseguir correr o AG até ao fim e avaliar se os resultados obtidos seriam melhores. Outra solução poderia passar pelo balanceamento do *dataset*, uniformizando o número de imagens por classe. Para além disso, seria interessant fazer também uma análise comparativa do benefício (ou não) da utilização de processos de *Image Augmentation*, avaliando o tempo de execução e os resultados obtidos.

Capítulo 7

Conclusão

A utilização de *Transfer Learning* na criação do modelo mostrou ser muito benéfica uma vez que diminui não só o tempo de treino como também resultou na obtenção de melhores resultados. Devido à limitação de tempo e recursos disponíveis não foi possível explorar mais arquiteturas de modelos e extender a procura pelos melhores hiperparâmetros. Porém, os resultados finais obtidos são bastante bons. O avaliação do modelo no dataset de teste obteve 97% de acurácia.

As possíveis causas encontradas para os valores obtidos são os parâmetros selecionados (função de otimização, função de ativação e *learning rate*), assim como o desbalanceamento do *dataset* de treino. Este *dataset* contém um número variado de imagens por espécie. No entanto, todas as espécies têm, pelo menos, 95 imagens. Este desbalanceamento pouco terá afetado o modelo, uma vez que conseguiu chegar aos 95% de acurácia de validação.

O facto de a acurácia não ser maior poderá dever-se ao facto da proporção de espécies macho para espécies fêmea. Cerca de 80% das imagens são masculinas e 20% femininas. Este desbalanceamento poderá estar a causar o não tão bom desempenho do modelo na validação.

A nível de trabalho futuro, exploraríamos mais a fundo as funções que definem o algoritmo genético, nomeadamente a função de mutação, assim como também o número de parâmetros a otimizar no AG. Analisaríamos também o eventual benefício da utilização de *Image Augmentation* e as razões que levaram aos resultados obtidos nas previsões.

Bibliografia

[1] Tian, Haiman & Chen, Shu-Ching & Shyu, Mei-Ling. (2019). Genetic Algorithm Based Deep Learning Model Selection for Visual Data Classification. 127-134. 10.1109/IRI.2019.00032.

[2] H. Tian, S. Pouyanfar, J. Chen, S. Chen and S. S. Iyengar, "Automatic Convolutional Neural Network Selection for Image Classification Using Genetic Algorithms,"2018 IEEE International Conference on Information Reuse and Integration (IRI), 2018, pp. 444-451, doi: 10.1109/IRI.2018.00071.

[3] Pere, Christophe. (2020, October 3), What is Image Classification? Data Augmentation? Transfer Learning?, *Towards Data Science*, https://towardsdatascience.com/what-is-image-classification-data-augmentation-transfer-learning-689389c3f6c8, acedido em: 25-4-2021

[4] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

[5] S. Gibb, H. M. La and S. Louis, "A Genetic Algorithm for Convolutional Network Structure Optimization for Concrete Crack Detection,"2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1-8, doi: 10.1109/CEC.2018.8477790.

[6] Loussaief, Sehla, and Afef Abdelkrim. "Convolutional neural network hyperparameters optimization based on genetic algorithms."International Journal of Advanced Computer Science and Applications 9.10 (2018): 252-266.

[7] A. Shukla, H. M. Pandey and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm,"2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015, pp. 515-519, doi: 10.1109/ABLAZE.2015.7154916.

[8] Lata, Saneh & Yadav, Saneh & Sohal, Asha. (2017). Comparative Study of Different Selection Techniques in Genetic Algorithm. International Journal of Engineering Science.

[9] Manmayi, Krishna. (2020, december), Hyper Parameter Optimization of CNN using genetic algorithm, *GitHub*, https://github.com/KrishnaManmayi/Hyper-Parameter-Optimization-of-CNN-using-genetic-algorithm/blob/main/Optimization_using_genetic_algorithm_Cifar10_using_CNN.ipynb, acedido em 25-4-2021