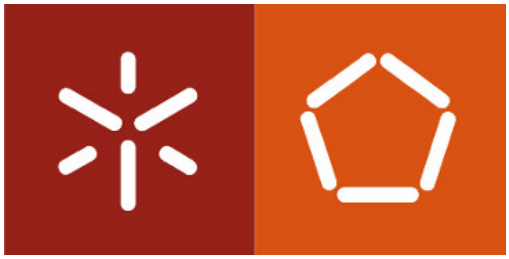



Universidade do Minho

Mestrado integrado em Engenharia Informática


# Trabalho Prático Individual 1


*João Nuno Cardoso Gonçalves de Abreu, A84802*


Computação Natural

4º Ano, 2º Semestre

Departmento de Informática

4 de maio de 2021

# Índice

# 1 Introdução

*Computer Vision* e *Neural Networks* são as novas tecnologias *IT* de técnicas de *machine learning*. Com os avanços das redes neuronais e a capacidade de ler imagens como *pixel density numbers*, várias empresas utilizam esta técnica para obter uma maior quantidade de dados.

Neste trabalho prático, pretende-se aplicar os conhecimentos leccionados ao longo da unidade curricular de Computação Natural para a classificação de aves através de imagens, fazendo uso de algoritmos CNN (Convolutional Neural Network). Noutras palavras, o projeto consiste em mecanismos que possibilitam a **preparação** do conjunto de dados necessários, seguido do **desenvolvimento** e **otimização** dos modelos de aprendizagem. Além disso, **Algoritmos Genéticos** foram aplicados para otimização automática da arquitetura CNN.

# 2 Preparação e Análise dos Dados

Este *dataset* é referente a um conjunto de imagens de aves, separadas em pastas sendo o critério de separação a espécie de cada ave. Como tal, o nosso objetivo será, dada uma imagem de uma ave nunca antes vista pelo algoritmo, classificá-la corretamente quanto à sua espécie.

As informações quanto ao *dataset* utilizado estarão presentes abaixo:

- Nº Espécies de Aves: 250;
- *Training Images*: 35215;
- *Validation Images*: 1250 (5 por espécie);
- *Test Images*: 1250 (5 por espécie);
- Tamanho das imagens: 224 x 224 x 3;
- Género das espécies: 80% das espécies são masculinas enquanto que os restantes 20% são femininas - o classificador pode ter um pior desempenho nas imagens com espécies femininas.

Figura 1: Exemplo de espécie de ave. (Annas Hummingbird)

É referido no enunciado que cada espécie teria, pelo menos, 100 *training images*, no entanto, isso não é a realidade pois há certas espécies com menos de 100 imagens para treino. Essas classes podem ser consultadas na Figura 2.

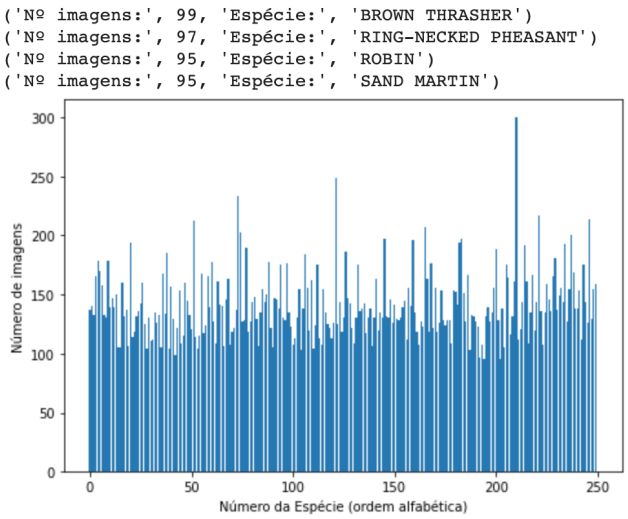



Figura 2: Espécies com menos de 100 imagens e número de imagens por espécie.

Uma vez que as aves já vinham corretamente separadas por espécie, não foi necessário qualquer alteração na importação dos dados, excepto a normalização da escala da imagem (1/255).

```
General_datagen = ImageDataGenerator(rescale=1./255)
```

# 3 Convolutional Neural Network Model

O modelo de CNN foi construído através da utilização da biblioteca *Keras* que fornece uma interface mais fácil e intuitiva para os utilizadores comparativamente a utilizar apenas *Tensorflow*.

## 3.1 Modelo para *hyperparameter-tuning*

Para o CNN desenvolvido, primeiramente optei por não usar nenhum modelo pré-treinado, para me familiarizar com as ferramentas e realizar alguns testes. Após algumas tentativas, reparei que valores como a *accuracy* e mesmo tempo de execução não estavam a ser propriamente aceitáveis e optei então por pesquisar por modelos pré-treinados. Deparei-me com um que acabou por ser o usado neste projeto, chamado ***MobileNet***. Tal como é referido em [1]:

> `MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parame -ters that efficiently trade off between latency and accurac -y. These hyper-parameters allow the model builder to choose the right sized model for their application based on the con -straints of the problem.`

Posto isto, apresentarei de seguida o modelo CNN desenvolvido:

```
base_mobilenet = MobileNet(
    weights = 'imagenet',
    include_top = False,
    input_shape = (224,224,3)
)
base_mobilenet.trainable = False # Freeze the mobilenet weights.

model = Sequential()
model.add(base_mobilenet)
model.add(Conv2D(f,(k,k),input_shape=(224,224,3),padding='same'))
```

```
model.add(Activation(a))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Flatten())
model.add(Dense(32*32))
model.add(Dropout(d))
model.add(Dense(250))
model.add(Activation('softmax'))

model.compile(
    Adam(l),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Como pode ser visto no código acima, existem variáveis não definidas, sendo estas [f,k,a,d,l]. Estas variáveis são os parâmetros ótimos que descobriremos com o uso do Algoritmo Genético mais à frente. Importante referir que o modelo apresentado foi baseado no modelo CNN da solução do exercício *Cats and Dogs CNN* realizado nas aulas práticas da unidade curricular com os devidos ajustes ao *dataset* em questão.

## 3.2 Modelo de Referência

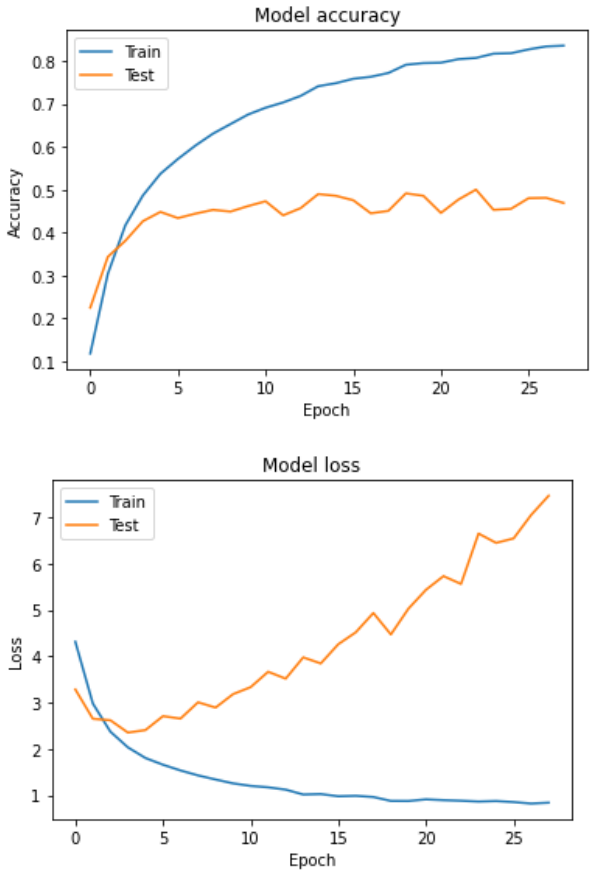
A razão pela qual criei um modelo de referência foi com o intuito de, depois, comparar os resultados obtidos da execução do Algoritmo Genético com um modelo sem qualquer otimização de hiper-parâmetros. Este modelo não difere em nada com o modelo adoptado no capítulo acima (pois perderia todo o sentido em compará-los) sendo que apenas difere nos **valores** das *layers*, valores estes que são os que o AG vai descobrir mais tarde como ótimos. Estes valores adoptados foram também baseados no modelo CNN da solução do exercício *Cats and Dogs CNN* realizado nas aulas práticas. O modelo de referência possui então a seguinte estrutura:

```
base_mobilenet = MobileNet(
```

```
    weights = 'imagenet',
    include_top = False,
    input_shape = SHAPE
)
base_mobilenet.trainable = False

model = Sequential()
model.add(base_mobilenet)
model.add(Conv2D(32,(3,3),input_shape=(224,224,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Flatten())
model.add(Dense(32*32))
model.add(Dense(250))
model.add(Activation('sigmoid'))
model.summary()

#Compile
model.compile(
    optimizer = 'adam',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

history = model.fit_generator(
    train_data,
    steps_per_epoch = Train_groups,
    epochs = 50,
    validation_data = validation_data,
    validation_steps = Valid_groups,
    verbose = 1,
    callbacks=[EarlyStopping(monitor = 'val_accuracy', patience = 5,
```

```
                                    restore_best_weights = True)])
```

A função *fitness* obteve os seguintes resultados:




```
40/40 [==============================] - 10s 248ms/step - loss: 5.4979 - accuracy: 0.4928
Test loss: 5.4978508949279785
Test accuracy: 0.4927999973297119
```

Figura 3: Modelo de referência.

Ou seja, para motivos de comparação mais tarde, temos uma *accuracy* $= 49\%$ e *loss* $= 5.49$.

# 4 Algoritmo Genético

Algoritmos Genéticos são um tipo de *learning algorithm* puramente inspirados pelo processo de evolução natural da natureza. Usam a ideia de que cruzar os pesos de duas boas redes neuronais resultaria numa rede neuronal melhor. A razão pela qual os algoritmos genéticos são tão eficazes é porque não existe um algoritmo de otimização direta, permitindo a possibilidade de obter resultados extremamente variados.

As suas vantagens são:

- **Computacionalmente não intensivo** - Não há cálculos de álgebra linear a serem feitos. Os únicos cálculos de *machine learning* necessários são passagens pelas redes neuronais.
- **Adaptável** - Pode-se adaptar e inserir muitos testes e maneiras diferentes de manipular a natureza flexível dos algoritmos genéticos.
- **Compreensível** - Para redes neuronais normais, os padrões de aprendizagem do algoritmo são enigmáticos, na melhor das hipóteses. Para algoritmos genéticos, é fácil entender porque algumas coisas acontecem: por exemplo, quando um algoritmo genético recebe o ambiente do jogo do galo, certas estratégias reconhecíveis desenvolvem-se lentamente. Esse é um grande benefício, pois o uso do *machine learning* é usar a tecnologia para nos ajudar a obter *insights* sobre assuntos importantes.

A desvantagem é que:

- **Demora muito tempo** - Maus *crossovers* e/ou *mutations* podem resultar num efeito negativo na precisão do programa e, portanto, tornar o programa mais lento para convergir ou atingir um certo limite de perda.

Agora vamos passar por alguns dos fundamentos do algoritmo genético.

- **Indivíduo** - Um indivíduo é a entidade que tenta resolver o problema dado.
- **Genes** - Conjunto de propriedades que caracterizam o indivíduo. Podem ser

um conjunto de strings ou, em nosso caso, os pesos da rede neuronal.

- **População** - Uma população é um conjunto de indivíduos que tentam superar um determinado problema.
- **Geração** - toda a população num determinado momento é a geração. Cada geração é melhor que a anterior. Cada indivíduo na geração atual é produzido a partir da última geração ou selecionado aleatoriamente a partir dela.
- **Elitismo** - A maioria dos indivíduos da elite da geração atual são capazes de se adaptar ao problema e, portanto, são promovidos diretamente para a próxima.
- **Acasalamento** - Da geração atual, os melhores indivíduos são escolhidos. Dois deles são escolhidos aleatoriamente e seus genes são misturados para formar um novo indivíduo.
- **Mutação** - os genes de um indivíduo recém-formado são modificados aleatoriamente para manter a aleatoriedade nas gerações.

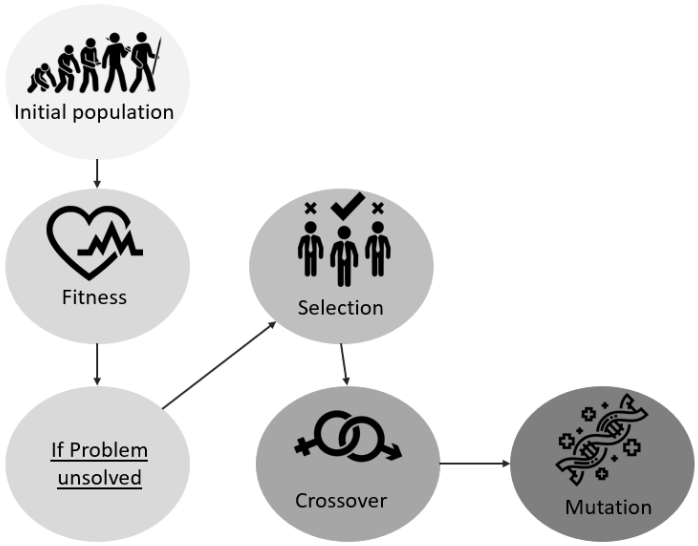



Figura 4: Fluxo do Algoritmo Genético.

## 4.1 Representação das Soluções

Uma solução terá de ter uma variedade de parâmetros (genes) relativos a possíveis parâmetros do modelo CNN a otimizar juntamente com os valores testados. Entre elas:

- Features map: [16, 32, 64, 128, 256]
- Kernel: [2, 3, 5, 7, 9]
- Activation: [Sigmoid, relu, tanh]
- Dropout: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
- Learning Rate: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]
- Fitness Value Loss
- Fitness Value Accuracy

Foi estipulada uma população de 6 soluções, sendo que o algoritmo irá parar ao fim de 10 execuções.

## 4.2 Função de Fitness

Imediatamente após a criação de cromossomas aleatórios, cada cromossoma realizará o treino CNN. Depois disso, cada valor de *accuracy* e *loss* será registado no final de cada cromossoma. O número de *epochs* usado foi 100 e steps_per_epoch foi 10.

## 4.3 Função de Seleção

Neste projeto é usado *Rank Selection* e *Tournament Selection*. Isto acontece porque os cromossomas primeiro precisam ser classificados antes de entrar no próximo processo. Após a classificação com base na menor *loss* e maior *accuracy*, metade dessa população será selecionada para realizar a *Tournament Selection*. Na *Tournament Selection*, os cromossomas são selecionados aleatoriamente, e apenas uma vez para serem selecionados para cada tournament. Isto significa que, quando o *Parent 1* está a selecionar o cromossoma A, é garantido que o *Parent*

*2* não possa selecionar o mesmo cromossoma. Isto é para produzir uma variedade de descendentes que não sejam iguais aos dos seus pais.

No entanto, foi adicionada uma regra que testava se, da metade dos cromossomas escolhidos para o *Tournament Selection*, todos estes possuíam *accuracy* $>$ 60% para não haver a necessidade de correr excessivas gerações uma vez que seria tempo desperdiçado ainda por cima adicionando *computing overhead*.

## 4.4 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 ***Single-Point Crossover***, onde, é escolhido um ponto nos cromossomas de ambos os pais e designado como ponto de cruzamento. Os bits à direita desse ponto são trocados entre os dois cromossomas pais. Isso resulta em dois filhos, cada um carregando algumas informações genéticas de ambos os pais.

## 4.5 Função de Mutação

Em termos de mutação definiu-se que 30% de probabilidade de ocorrer era equilibrado, resultando em, por cada 10 soluções, 3 iriam sofrer mutação. Esta ocorre em apenas num dos genes que possui, tendo todas estas a mesma probabilidade de ocorrer.

# 5 Resultados

Neste capítulo apresentarei os resultados obtidos após executar o Algoritmo Genético duas vezes. A razão por detrás desta decisão deve-se ao facto de não considerar satisfatórios os resultados obtidos, por isso, depois de alguns ajustes feitos ao modelo, foi feita uma nova execução. Ambas estarão presentes no relatório, juntamente com os ajustes referidos.

## 5.1 Primeira execução do Algoritmo Genético

### 5.1.1 Resultados AG

Dado as opções iniciais dos parâmetros definidos em 4.1, os resultados obtidos do Algoritmo Genético para *hyperparameter-tuning* foram os seguintes:

```
Estrutura:
[Features Map, Kernel, Activation, Dropout, L Rate, Loss, Accuracy]

Geração 0 foi:
[64, 2, 'sigmoid', 0.5, 0.001, 0.0, 0.0]
[256, 2, 'tanh', 0.5, 0.001, 0.0, 0.0]
[256, 9, 'tanh', 0.0, 0.5, 0.0, 0.0]
[256, 5, 'tanh', 0.3, 0.1, 0.0, 0.0]
[32, 9, 'relu', 0.3, 0.01, 0.0, 0.0]
[128, 7, 'sigmoid', 0.2, 0.1, 0.0, 0.0]

Geração 10 foi:
[16, 2, 'tanh', 0.5, 0.001, 2.6004033851623536, 0.409781250001397]
[16, 2, 'tanh', 0.5, 0.001, 2.613929785490036, 0.40974999997066336]
[16, 2, 'tanh', 0.5, 0.001, 2.614882788658142, 0.4069167682295665]
[16, 2, 'tanh', 0.5, 0.1, 2.6004033851623536, 0.409781250001397]
[16, 2, 'tanh', 0.5, 0.01, 2.613929785490036, 0.40974999997066336]
[16, 2, 'tanh', 0.5, 0.01, 2.613929785490036, 0.40974999997066336]
```

Daqui podemos concluir que, dado os parâmetros iniciais aleatoriamente escolhidos, o algoritmo converge para um resultado com *loss* = 2.6 e *accuracy* = 40%. Uma das primeiras questões que se pode fazer a estes resultados seria de onde vem o valor do *features map* = 16 na geração final se nunca aparece na primeira geração? Isso deve-se ao facto de, na geração seguinte, na geração 1, ter ocorrido uma mutação no primeiro bit que acabou por se manter até ao final. O output de todo o Algoritmo Genético no *notebook* encontra-se um bocado desorganizado devido ao excesso de linhas que indicam o progresso das *epochs*, por isso, foi criado um ficheiro à parte chamado *output-ga-clean-1st-run.txt*. Mais conclusões acerca

dos resultados serão explicados em 5.1.3.

### 5.1.2 Resultados CNN

Após encontrados os parâmetros ótimos a partir do Algoritmo Genético, foi corrida a função de *fitness* **de novo** com os novos valores da seguinte maneira.

```
base_mobilenet = MobileNet(
    weights = 'imagenet',
    include_top = False,
    input_shape = SHAPE
)
base_mobilenet.trainable = False

model = Sequential()
model.add(base_mobilenet)
model.add(Conv2D(16,(2,2),input_shape=(224,224,3),padding='same'))
model.add(Activation('tanh'))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Flatten())
model.add(Dense(32*32))
model.add(Dropout(0.5))
model.add(Dense(250))
model.add(Activation('softmax'))
model.summary()

model.compile(
    Adam(0.01),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

Ou seja, os valores usados foram:

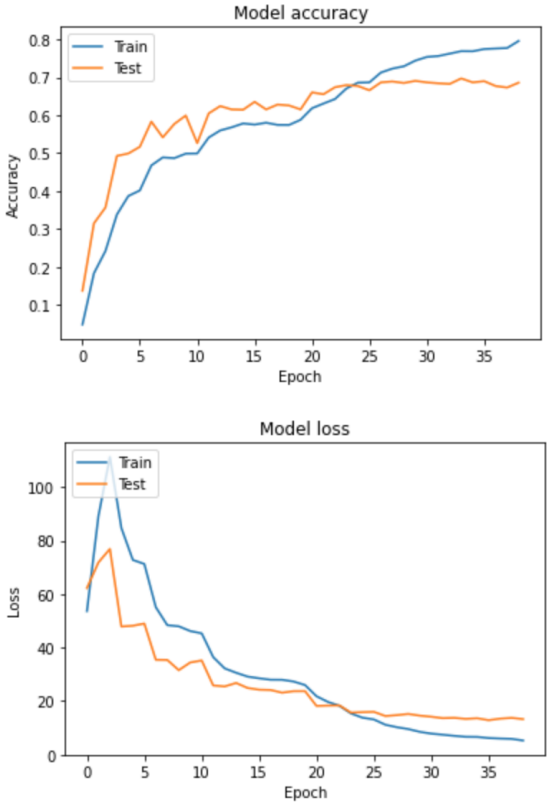
- Features map: 16

- Kernel: 2
- Activation: tanh
- Dropout: 0.5
- Learning Rate: 0.01

E a função de fitness usada foi:

```
model.fit_generator(
    train_data,
    steps_per_epoch = Train_groups,
    epochs = 50,
    validation_data = validation_data,
    validation_steps = Valid_groups,
    verbose = 1,
    callbacks=[EarlyStopping(monitor = 'val_accuracy', patience = 5,
                        restore_best_weights = True),
           ReduceLROnPlateau(monitor = 'val_loss', factor = 0.7,
                            patience = 2, verbose = 1)
    ])
```

Os resultados obtidos foram os seguintes:




```
40/40 [==============================] - 24s 606ms/step - loss: 11.6598 - accuracy: 0.6968
Test loss: 11.659765243530273
Test accuracy: 0.6967999935150146
```

Figura 5: Resultados CNN.

### 5.1.3 Conclusões sobre os Resultados

A partir dos resultados em 5.1.1 e em 5.1.2 conseguimos concluir:

- O que difere entre os cromossomas resultantes do AG foi apenas o LR, sendo assim escolhi o que tinha maior *accuracy* (que é quase insignificante) para correr de novo a função de *fitness* do modelo CNN.
- No entanto, na função *fitness* do modelo CNN que corri **após** obter os parâmetros resultantes do AG, coloquei uma nova *callback* que não tinha usado até agora chamada ***ReduceLROnPlateau*** na função de *fitness* que vai reduzindo o LR quando a *accuracy* para de melhorar. O LR passou de 0.01 -> 0.0005. É uma grande diferença mas podia ter usado um dos outros cromos-

somas resultantes (que têm todos praticamente os mesmos valores de *loss* e *accuracy*) com LR de 0.001 que a diferença já seria menor.

- Ou seja, na função *fitness* do modelo CNN fim do Algoritmo Genético a *accuracy* era 40%. Depois, ao pegar nos parâmetros resultantes do AG e correndo a função fitness de novo com a *callback* referida anteriormente, a *accuracy* passou para cerca de 70%.
- Também consigo concluir que, apesar de ter colocado valores como 0.0005 nas possibilidades para as combinações dos parâmetros, este não foi escolhido pelo algoritmo inicialmente, o que leva a pensar que, dada outra combinação de valores iniciais, o meu resultado poderia ter sido melhor, não só para o LR como para todos os outros parâmetros.

## 5.2 Segunda execução do Algoritmo Genético

Como já foi referido anteriormente, decidi executar o Algoritmo Genético mais uma vez na busca de melhores resultados. Os ajustes feitos foram no intervalo de possibilidades do *Learning Rate*, que passaram de [0.0001,0.0005, 0.001, 0.005, 0.01,0.05,0.1,0.5] para [0.0001,0.0005, 0.001, 0.005, 0.01], ou seja, retirei os 2 valores mais altos da lista, uma vez que na execução anterior do AG, a solução ótima possuía um LR bastante mais baixo do que 0.1 e 0.5. O número de gerações também foi reduzido para metade pois o AG já tinha convergido bastante mais cedo, sendo que agora passaremos a ter 5 gerações mas com os mesmos 6 indivíduos na população.

### 5.2.1 Resultados AG

Os resultados desta vez foram:

```
Geração 0 foi:
[64, 7, 'tanh', 0.2, 0.0001, 0.0, 0.0]
[128, 2, 'tanh', 0.4, 0.01, 0.0, 0.0]
[16, 5, 'tanh', 0.0, 0.01, 0.0, 0.0]
[32, 3, 'tanh', 0.2, 0.01, 0.0, 0.0]
```

```
[128, 5, 'relu', 0.1, 0.0001, 0.0, 0.0]
[32, 2, 'sigmoid', 0.4, 0.01, 0.0, 0.0]

Selected Ranked Chromosomes:
[64, 7, 'tanh', 0.1, 0.0001, 1.8223397767543792, 0.6331485137715935]
[64, 7, 'tanh', 0.1, 0.0001, 1.8903586041927338, 0.6210340339806862]
[128, 5, 'relu', 0.1, 0.0001, 1.8501557099819184, 0.6012701092893258]
```

Desta vez o *output* do algoritmo não mostra a população final mas sim metade da população, metade esta que satisfez a regra enunciada em 4.3. Ou seja, o algoritmo parou antecipadamente, sendo neste caso na 4ª geração. Também estará presente num ficheiro chamado *output-ga-clean-2nd-run.txt* o output mais organizado produzido da execução do AG. Agora, possuímos *accuracy* = 0.63% e *loss* = 1.82.

### 5.2.2 Resultados CNN

Após encontrados os parâmetros ótimos a partir do Algoritmo Genético, foi corrida a função de *fitness* **de novo** com os novos valores da seguinte maneira.

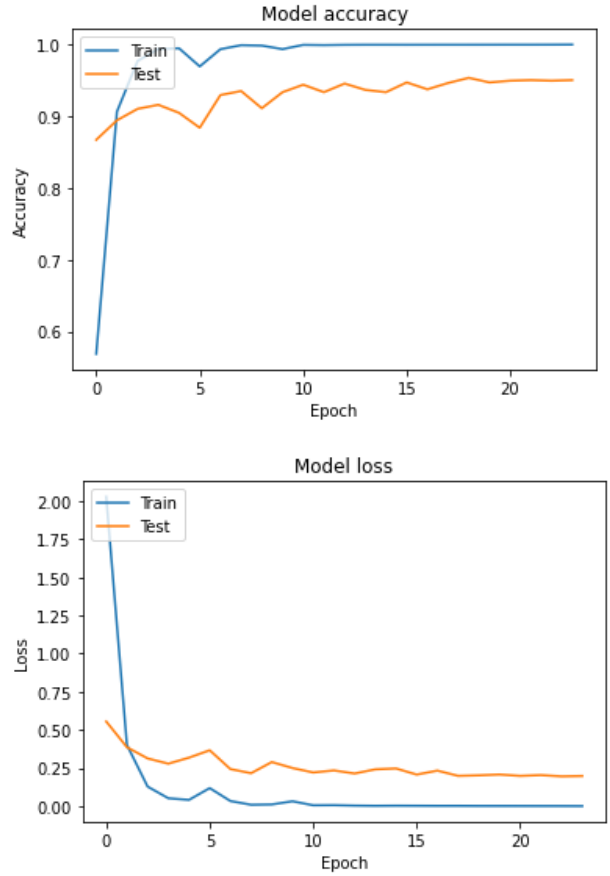
```
base_mobilenet = MobileNet(
    weights = 'imagenet',
    include_top = False,
    input_shape = SHAPE
)
base_mobilenet.trainable = False

model = Sequential()
model.add(base_mobilenet)
model.add(Conv2D(64,(7,7),input_shape=(224,224,3),padding='same'))
model.add(Activation('tanh'))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Flatten())
model.add(Dense(32*32))
```

```
model.add(Dropout(0.1))
model.add(Dense(250))
model.add(Activation('softmax'))
model.summary()

model.compile(
    Adam(0.0001),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

Ou seja, os valores usados foram do cromossoma com melhor *accuracy*, ou seja:

- Features map: 64
- Kernel: 7
- Activation: tanh
- Dropout: 0.1
- Learning Rate: 0.0001

A função de fitness usada não foi alterada. Os resultados obtidos foram os seguintes:




```
40/40 [==============================] - 13s 312ms/step - loss: 0.1216 - accuracy: 0.9656
Test loss: 0.12158472836017609
Test accuracy: 0.9656000137329102
```

Figura 6: Resultados CNN.

### 5.2.3 Conclusões sobre os Resultados

Desta vez, após os devidos ajustes feitos e com um pouco de mais sorte, conseguimos passar duma *accuracy* = 70% para *accuracy* = 96%, tratando-se de uma melhoria bastante significativa. Relembro que antes de sequer ter sido executado o Algoritmo Genético, a *accuracy* = 49%, podendo assim concluir que o uso deste tipo de algoritmo teve bastante impacto nos resultados de uma forma positiva, tanto na primeira como na segunda tentativa de execução.

# 6 Conclusões

Algoritmos 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 aprendi que dada as tantas possibilidades de arquiteturas possíveis para *Convolutional Neural Networks*, escolher manualmente uma é dispendioso e não praticado, sendo que me foquei pelas arquiteturas tradicionais, mas usando uma ferramenta por trás como as AG's este processo torna-se simples e com pouco esforço. É importante referir que a solução encontrada é apenas uma de muitas boas soluções que a AG encontrou, sendo que é possível existirem muitas mais ainda nesta nuvem de soluções.

Aspetos a melhorar no trabalho seria a tomada em atenção do facto de existirem muito mais imagens de espécies masculinas do que femininas para uma melhor *accuracy* do modelo. A solução que proporia seria fazer uso de *data augmentation* ou utilizando, por exemplo, *grey-scale evaluation* para tornar mais fácil prever as aves femininas uma vez que as suas cores são menos intensas do que as masculinas.

# Referências

[1] *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, `https://arxiv.org/abs/1704.04861`, Acedido: 02-05-2021.