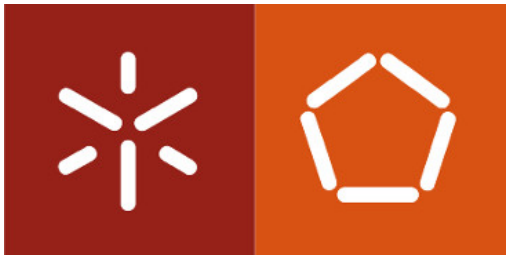


Universidade do Minho

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

3 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** devem ser aplicados para otimização automática da arquitetura CNN. Quanto aos dados presentes neste relatório como imagens, gráficos, conjunto de dados, etc, estes foram obtidos usando o notebook chamado *auxiliar-relatório.ipynb* para apenas a secção 2, enquanto que os dados presentes na secção 5.2 e 6 encontram-se nos notebooks *CNN-with-GA.ipynb* e *CNN-without-GA.ipynb*, respetivamente.

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.

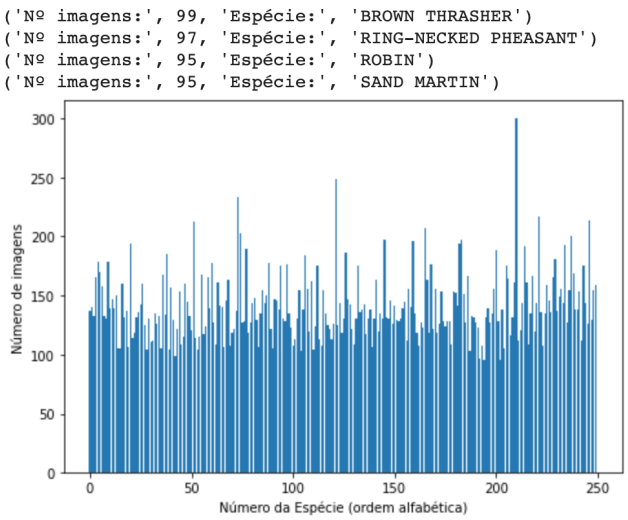


Figura 2: Espécies com menos de 100 imagens e número de imagens por espécie.

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*.

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 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.

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.

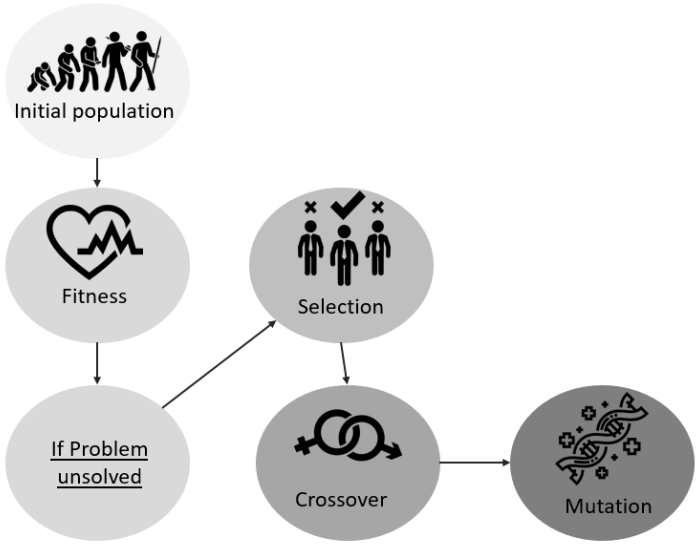


Figura 3: Fluxo do Algoritmo Genético.

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.

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

5.1 Resultados GA

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.txt* onde se pode analisar casos de mutações como este, ficheiro este que também estará presente no capítulo Anexos [8]. Mais conclusões acerca dos resultados serão explicados em 5.3.

5.2 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']
)
```

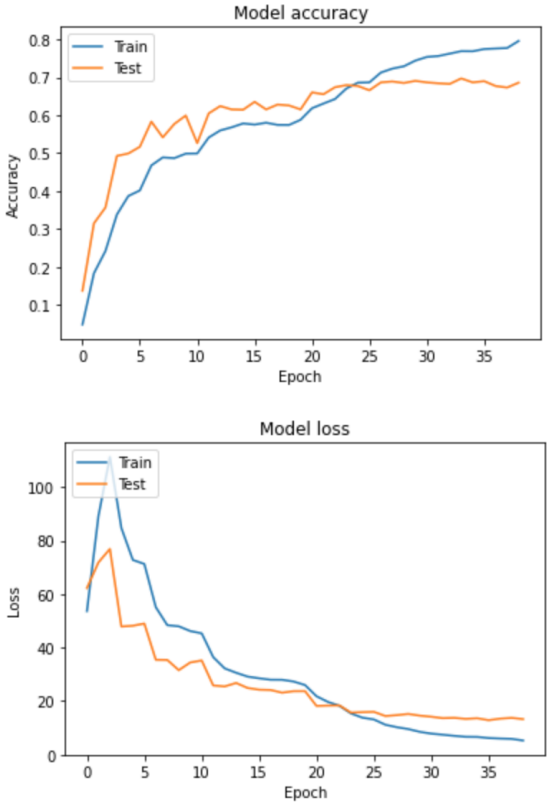
Ou seja, os valores usados foram:

- 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 4: Resultados CNN.

5.3 Conclusões sobre os Resultados

A partir dos resultados em 5.1 e em 5.2 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 cromossomas 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.

6 Extra

Após concluído o trabalho, pesquisei mais sobre *MobileNet* e encontrei o seguinte modelo:

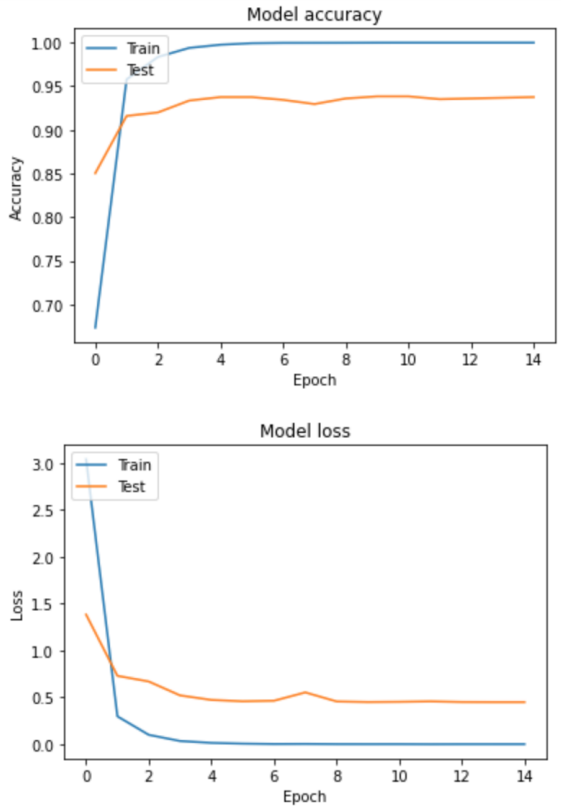
```
base_mobilenet = MobileNet(
    weights = 'imagenet',
    include_top = False,
    input_shape = SHAPE
)
base_mobilenet.trainable = False

model = Sequential()
model.add(base_mobilenet)

model.add(Flatten())
model.add(Activation('relu'))
model.add(Dense(250))
model.add(Activation('softmax'))
```

```
model.compile(
    optimizer = tf.keras.optimizers.SGD(
        lr=0.001,
        momentum=0.9,
        nesterov=True
    ),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

É um modelo que faz uso de poucas *layers* e que usa na função *compile* um otimizador diferente chamado *SGD*. Usando este modelo e usando a função de *fitness* referida anteriormente, os resultados foram excelentes.




```
40/40 [==============================] - 10s 243ms/step - loss: 0.2751 - accuracy: 0.9584
Test loss: 0.27506011724472046
Test accuracy: 0.9584000110626221
```

Figura 5: Resultados CNN Novo Modelo.

Com 95.8% de *accuracy*, este modelo classifica corretamente 1198/1250 aves! Daqui posso concluir que o resultado obtido pelo Algoritmo Genético ficou bastante a desejar em comparação com o que realmente pode ser alcançado, sendo que a razão pode ter sido no tipo e valores das *layers* usadas para o *hyperparameter-tuning* do AG.

7 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 GA'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 GA encontrou, sendo que é possível existirem muitas mais ainda nesta nuvem de soluções. Uma nova execução de todo o Algoritmo Genético com melhores valores iniciais poderia ter melhorado substancialmente os resultados obtidos, mas devido à quantidade de tempo que este processo exige e à escassez do mesmo, infelizmente, não foi possível realizar novos testes.

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.

8 Anexos

8.1 Output Algoritmo Genético

```
Generation 0
[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]
------------------------------------------------------------
64 2 sigmoid 0.5 0.001
accuracy = 0.2451716174534522
loss = 3.5416400349140167
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.3894104788010009
loss = 5.28730188369751
------------------------------------------------------------
256 9 tanh 0.0 0.5
accuracy = 0.004593750096391886
loss = 7137148.51
------------------------------------------------------------
256 5 tanh 0.3 0.1
accuracy = 0.005000000102445484
loss = 88517.8793359375
------------------------------------------------------------
32 9 relu 0.3 0.01
accuracy = 0.004281250077765435
loss = 5.93030553817749
------------------------------------------------------------
128 7 sigmoid 0.2 0.1
```

```
accuracy = 0.0038767533795908094
loss = 2403.910565185547
------------------------------------------------------------
Selected Ranked Chromosomes:
[256, 2, 'tanh', 0.5, 0.001, 5.28730188369751, 0.3894104788010009]
[64, 2, 'sigmoid', 0.5, 0.001, 3.5416400349140167, 0.2451716174534522]
[256, 5, 'tanh', 0.3, 0.1, 88517.8793359375, 0.005000000102445484]

Selected Childs after crossover:
Child 1 = [64, 2, 'sigmoid', 0.5, 0.001, 5.28730188369751, 0.3894104788010009]
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 3.5416400349140167, 0.2451716174534522]

rate: 0.015004846315720743 < 0.3 -> Mutation Time!
index: 4

New Childs in Generation 1:
[64, 2, 'sigmoid', 0.5, 0.05, 5.28730188369751, 0.3894104788010009]
[256, 2, 'tanh', 0.5, 0.001, 3.5416400349140167, 0.2451716174534522]

New Generation 1:
[256, 2, 'tanh', 0.5, 0.001, 5.28730188369751, 0.3894104788010009]
[64, 2, 'sigmoid', 0.5, 0.001, 3.5416400349140167, 0.2451716174534522]
[256, 5, 'tanh', 0.3, 0.1, 88517.8793359375, 0.005000000102445484]
[64, 2, 'sigmoid', 0.5, 0.05, 5.28730188369751, 0.3894104788010009]
[256, 2, 'tanh', 0.5, 0.001, 3.5416400349140167, 0.2451716174534522]
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.40447122694924476
loss = 5.062942066192627
------------------------------------------------------------
64 2 sigmoid 0.5 0.001
accuracy = 0.22585602284409106
```

```
loss = 3.6712683379650115
------------------------------------------------------------
256 5 tanh 0.3 0.1
accuracy = 0.0061632633139379325
loss = 85547.86375
------------------------------------------------------------
64 2 sigmoid 0.5 0.05
accuracy = 0.004187500085681677
loss = 8.040920476913453
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.4008868613280356
loss = 5.168866183757782
------------------------------------------------------------
Selected Ranked Chromosomes:
[256, 2, 'tanh', 0.5, 0.001, 5.062942066192627, 0.40447122694924476]
[256, 2, 'tanh', 0.5, 0.001, 5.168866183757782, 0.4008868613280356]
[64, 2, 'sigmoid', 0.5, 0.001, 3.6712683379650115, 0.22585602284409106]

Selected Childs after crossover:
Child 1 = [64, 2, 'sigmoid', 0.5, 0.001, 5.168866183757782, 0.4008868613280356]
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 3.6712683379650115, 0.22585602284409106]

rate: 0.14755298740441547 < 0.3 -> Mutation Time!
index: 0

New Childs in Generation 2:
[128, 2, 'sigmoid', 0.5, 0.001, 5.168866183757782, 0.4008868613280356]
[16, 2, 'tanh', 0.5, 0.001, 3.6712683379650115, 0.22585602284409106]

New Generation 2:
[256, 2, 'tanh', 0.5, 0.001, 5.062942066192627, 0.40447122694924476]
```

```
[256, 2, 'tanh', 0.5, 0.001, 5.168866183757782, 0.4008868613280356]
[64, 2, 'sigmoid', 0.5, 0.001, 3.6712683379650115, 0.22585602284409106]
[128, 2, 'sigmoid', 0.5, 0.001, 5.168866183757782, 0.4008868613280356]
[16, 2, 'tanh', 0.5, 0.001, 3.6712683379650115, 0.22585602284409106]
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.390437500202097
loss = 5.243131053447724
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.39575824967585504
loss = 5.215166244506836
------------------------------------------------------------
64 2 sigmoid 0.5 0.001
accuracy = 0.24781250092433765
loss = 3.5369154167175294
------------------------------------------------------------
128 2 sigmoid 0.5 0.001
accuracy = 0.22225000043399631
loss = 3.7138738870620727
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.40775979732628914
loss = 2.6197396981716157
------------------------------------------------------------
Selected Ranked Chromosomes:
[16, 2, 'tanh', 0.5, 0.001, 2.6197396981716157, 0.40775979732628914]
[256, 2, 'tanh', 0.5, 0.001, 5.215166244506836, 0.39575824967585504]
[256, 2, 'tanh', 0.5, 0.001, 5.243131053447724, 0.390437500202097]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 5.215166244506836, 0.39575824967585504]
```

```
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 2.6197396981716157, 0.40775979732628914]

rate: 0.5590084587665586
New Childs in Generation 3:
[16, 2, 'tanh', 0.5, 0.001, 5.215166244506836, 0.39575824967585504]

[256, 2, 'tanh', 0.5, 0.001, 2.6197396981716157, 0.40775979732628914]

New Generation 3:
[16, 2, 'tanh', 0.5, 0.001, 2.6197396981716157, 0.40775979732628914]
[256, 2, 'tanh', 0.5, 0.001, 5.215166244506836, 0.39575824967585504]
[256, 2, 'tanh', 0.5, 0.001, 5.243131053447724, 0.390437500202097]
[16, 2, 'tanh', 0.5, 0.001, 5.215166244506836, 0.39575824967585504]
[256, 2, 'tanh', 0.5, 0.001, 2.6197396981716157, 0.40775979732628914]
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.39071833695750685
loss = 2.717617791891098
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.39768605640158056
loss = 5.201920580863953
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.40593749966472387
loss = 5.095706262588501
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.3997030736738816
loss = 2.662250417470932
------------------------------------------------------------
256 2 tanh 0.5 0.001
```

```
accuracy = 0.4019750428223051
loss = 5.1303731322288515
------------------------------------------------------------
Selected Ranked Chromosomes:
[256, 2, 'tanh', 0.5, 0.001, 5.095706262588501, 0.40593749966472387]
[256, 2, 'tanh', 0.5, 0.001, 5.1303731322288515, 0.4019750428223051]
[16, 2, 'tanh', 0.5, 0.001, 2.662250417470932, 0.3997030736738816]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 5.095706262588501, 0.40593749966472387]
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 2.662250417470932, 0.3997030736738816]

rate: 0.23524477242234132 < 0.3 -> Mutation Time!
index: 5

New Childs in Generation 4:
[16, 2, 'tanh', 0.5, 0.001, 5.095706262588501, 0.40593749966472387]
[256, 2, 'tanh', 0.5, 0.001, 2.662250417470932, 0.3997030736738816]

New Generation 4:
[256, 2, 'tanh', 0.5, 0.001, 5.095706262588501, 0.40593749966472387]
[256, 2, 'tanh', 0.5, 0.001, 5.1303731322288515, 0.4019750428223051]
[16, 2, 'tanh', 0.5, 0.001, 2.662250417470932, 0.3997030736738816]
[16, 2, 'tanh', 0.5, 0.001, 5.095706262588501, 0.40593749966472387]
[256, 2, 'tanh', 0.5, 0.001, 2.662250417470932, 0.3997030736738816]
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.39596875275718046
loss = 5.203267204761505
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.40199999829754235
```

```
loss = 5.147877731323242
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.40796875227242707
loss = 2.6156194722652435
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4163749985792674
loss = 2.563546588420868
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.40673628270626067
loss = 5.047762641906738
------------------------------------------------------------
Selected Ranked Chromosomes:
[16, 2, 'tanh', 0.5, 0.001, 2.563546588420868, 0.4163749985792674]
[16, 2, 'tanh', 0.5, 0.001, 2.6156194722652435, 0.40796875227242707]
[256, 2, 'tanh', 0.5, 0.001, 5.047762641906738, 0.40673628270626067]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 2.6156194722652435, 0.40796875227242707]
Child 2 = [16, 2, 'tanh', 0.5, 0.001, 2.563546588420868, 0.4163749985792674]

rate: 0.11015831424059874 < 0.3 -> Mutation Time!
index: 1

New Childs in Generation 5:
[16, 7, 'tanh', 0.5, 0.001, 2.6156194722652435, 0.40796875227242707]
[16, 5, 'tanh', 0.5, 0.001, 2.563546588420868, 0.4163749985792674]

New Generation 5:
[16, 2, 'tanh', 0.5, 0.001, 2.563546588420868, 0.4163749985792674]
```

```
[16, 2, 'tanh', 0.5, 0.001, 2.6156194722652435, 0.40796875227242707]
[256, 2, 'tanh', 0.5, 0.001, 5.047762641906738, 0.40673628270626067]
[16, 7, 'tanh', 0.5, 0.001, 2.6156194722652435, 0.40796875227242707]
[16, 5, 'tanh', 0.5, 0.001, 2.563546588420868, 0.4163749985792674]
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.407553628932219
loss = 2.6074810576438905
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.39215625148965044
loss = 2.7071132230758668
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.40830930343829097
loss = 5.040172629356384
------------------------------------------------------------
16 7 tanh 0.5 0.001
accuracy = 0.15824999956879765
loss = 4.065364103317261
------------------------------------------------------------
16 5 tanh 0.5 0.001
accuracy = 0.22848989305784925
loss = 3.5979675030708314
------------------------------------------------------------
Selected Ranked Chromosomes:
[256, 2, 'tanh', 0.5, 0.001, 5.040172629356384, 0.40830930343829097]
[16, 2, 'tanh', 0.5, 0.001, 2.6074810576438905, 0.407553628932219]
[16, 2, 'tanh', 0.5, 0.001, 2.7071132230758668, 0.39215625148965044]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 5.040172629356384, 0.40830930343829097]
```

```
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 2.6074810576438905, 0.407553628932219]

rate: 0.08548794253061287 < 0.3 -> Mutation Time!
index: 5

New Childs in Generation 6:
[16, 2, 'tanh', 0.5, 0.001, 5.040172629356384, 0.40830930343829097]
[256, 2, 'tanh', 0.5, 0.001, 2.6074810576438905, 0.407553628932219]

New Generation 6:
[256, 2, 'tanh', 0.5, 0.001, 5.040172629356384, 0.40830930343829097]
[16, 2, 'tanh', 0.5, 0.001, 2.6074810576438905, 0.407553628932219]
[16, 2, 'tanh', 0.5, 0.001, 2.7071132230758668, 0.39215625148965044]
[16, 2, 'tanh', 0.5, 0.001, 5.040172629356384, 0.40830930343829097]
[256, 2, 'tanh', 0.5, 0.001, 2.6074810576438905, 0.407553628932219]
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.39914727716939524
loss = 5.205491037368774
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4106233513285406
loss = 2.6015638864040374
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.40037500103004275
loss = 2.6415662240982054
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.3973749997327104
loss = 2.6392199087142942
------------------------------------------------------------
```

```
256 2 tanh 0.5 0.001
accuracy = 0.4036917288508266
loss = 5.1233360075950625
------------------------------------------------------------
Selected Ranked Chromosomes:
[16, 2, 'tanh', 0.5, 0.001, 2.6015638864040374, 0.4106233513285406]
[256, 2, 'tanh', 0.5, 0.001, 5.1233360075950625, 0.4036917288508266]
[16, 2, 'tanh', 0.5, 0.001, 2.6415662240982054, 0.40037500103004275]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 5.1233360075950625, 0.4036917288508266]
Child 2 = [256, 2, 'tanh', 0.5, 0.001, 2.6015638864040374, 0.4106233513285406]

rate: 0.43823204145233685
New Childs in Generation 67:
[16, 2, 'tanh', 0.5, 0.001, 5.1233360075950625, 0.4036917288508266]

[256, 2, 'tanh', 0.5, 0.001, 2.6015638864040374, 0.4106233513285406]

New Generation 7:
[16, 2, 'tanh', 0.5, 0.001, 2.6015638864040374, 0.4106233513285406]
[256, 2, 'tanh', 0.5, 0.001, 5.1233360075950625, 0.4036917288508266]
[16, 2, 'tanh', 0.5, 0.001, 2.6415662240982054, 0.40037500103004275]
[16, 2, 'tanh', 0.5, 0.001, 5.1233360075950625, 0.4036917288508266]
[256, 2, 'tanh', 0.5, 0.001, 2.6015638864040374, 0.4106233513285406]
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.40968678024364635
loss = 2.607346225976944
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.4035187710565515
```

```
loss = 5.147452301979065
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4049687500996515
loss = 2.647631930112839
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4032130773505196
loss = 2.6485305964946746
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.3927499990561046
loss = 5.286952610015869
------------------------------------------------------------
Selected Ranked Chromosomes:
[16, 2, 'tanh', 0.5, 0.001, 2.607346225976944, 0.40968678024364635]
[16, 2, 'tanh', 0.5, 0.001, 2.647631930112839, 0.4049687500996515]
[256, 2, 'tanh', 0.5, 0.001, 5.147452301979065, 0.4035187710565515]

Selected Childs after crossover:
Child 1 = [256, 2, 'tanh', 0.5, 0.001, 2.607346225976944, 0.40968678024364635]
Child 2 = [16, 2, 'tanh', 0.5, 0.001, 5.147452301979065, 0.4035187710565515]

rate: 0.3940514637032848
New Childs in Generation 8:
[256, 2, 'tanh', 0.5, 0.001, 2.607346225976944, 0.40968678024364635]

[16, 2, 'tanh', 0.5, 0.001, 5.147452301979065, 0.4035187710565515]

New Generation 8:
[16, 2, 'tanh', 0.5, 0.001, 2.607346225976944, 0.40968678024364635]
[16, 2, 'tanh', 0.5, 0.001, 2.647631930112839, 0.4049687500996515]
```

```
[256, 2, 'tanh', 0.5, 0.001, 5.147452301979065, 0.4035187710565515]
[256, 2, 'tanh', 0.5, 0.001, 2.607346225976944, 0.40968678024364635]
[16, 2, 'tanh', 0.5, 0.001, 5.147452301979065, 0.4035187710565515]
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4097442247485742
loss = 2.6053827202320097
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4035000028088689
loss = 2.6073183953762054
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.3999062504991889
loss = 5.133699676990509
------------------------------------------------------------
256 2 tanh 0.5 0.001
accuracy = 0.3933437493816018
loss = 5.290406768321991
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4126263420144096
loss = 2.596596825122833
------------------------------------------------------------
Selected Ranked Chromosomes:
[16, 2, 'tanh', 0.5, 0.001, 2.596596825122833, 0.4126263420144096]
[16, 2, 'tanh', 0.5, 0.001, 2.6053827202320097, 0.4097442247485742]
[16, 2, 'tanh', 0.5, 0.001, 2.6073183953762054, 0.4035000028088689]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 2.596596825122833, 0.4126263420144096]
Child 2 = [16, 2, 'tanh', 0.5, 0.001, 2.6053827202320097, 0.4097442247485742]
```

```
rate: 0.983765771355859
New Childs in Generation 9:
[16, 2, 'tanh', 0.5, 0.001, 2.596596825122833, 0.4126263420144096]

[16, 2, 'tanh', 0.5, 0.001, 2.6053827202320097, 0.4097442247485742]

New Generation 9:
[16, 2, 'tanh', 0.5, 0.001, 2.596596825122833, 0.4126263420144096]
[16, 2, 'tanh', 0.5, 0.001, 2.6053827202320097, 0.4097442247485742]
[16, 2, 'tanh', 0.5, 0.001, 2.6073183953762054, 0.4035000028088689]
[16, 2, 'tanh', 0.5, 0.001, 2.596596825122833, 0.4126263420144096]
[16, 2, 'tanh', 0.5, 0.001, 2.6053827202320097, 0.4097442247485742]
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4045601277635433
loss = 2.621526962518692
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.40974999997066336
loss = 2.613929785490036
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4015624982630834
loss = 2.651753945350647
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.4069167682295665
loss = 2.614882788658142
------------------------------------------------------------
16 2 tanh 0.5 0.001
accuracy = 0.409781250001397
```

```
loss = 2.6004033851623536
------------------------------------------------------------
Selected Ranked Chromosomes:
[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]

Selected Childs after crossover:
Child 1 = [16, 2, 'tanh', 0.5, 0.001, 2.6004033851623536, 0.409781250001397]
Child 2 = [16, 2, 'tanh', 0.5, 0.001, 2.613929785490036, 0.40974999997066336]

rate: 0.02105898445900245 < 0.3 -> Mutation Time!
index: 4

New Childs in Generation 10:
[16, 2, 'tanh', 0.5, 0.1, 2.6004033851623536, 0.409781250001397]
[16, 2, 'tanh', 0.5, 0.01, 2.613929785490036, 0.40974999997066336]

New Generation 10:
[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]
------------------------------------------------------------
```

Referências

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