



**UNIVERSIDADE DO MINHO**

Computação Natural

**TP1 - Convolutional Neural Networks on Image  
Classification**

Rafael Lourenço  
(A86266)

# Conteúdo

<b>1</b>	<b>Introdução e Contextualização</b>	<b>2</b>
<b>2</b>	<b>Dataset</b>	<b>3</b>
2.1	Descrição do Dataset . . . . .	3
2.2	Data Augmentation . . . . .	3
2.3	Keras pre-processing . . . . .	4
2.4	Batch size . . . . .	4
<b>3</b>	<b>Convolutional neural network</b>	<b>5</b>
3.1	Covolution2D Layer . . . . .	5
3.1.1	Filters . . . . .	5
3.1.2	Kernel . . . . .	6
3.1.3	Padding . . . . .	6
3.1.4	Activation Function . . . . .	6
3.1.5	Limitações de uma <b>conv2D</b> . . . . .	7
3.2	Pooling Layer . . . . .	7
3.3	Dropout Layer . . . . .	7
3.4	Fully Connected Layer . . . . .	7
3.5	Callbacks . . . . .	8
3.6	Modelos Criados . . . . .	8
3.6.1	Modelo Custom 1 . . . . .	8
3.6.2	Modelo Custom 2 . . . . .	9
3.6.3	Modelo Custom 3 . . . . .	11
3.6.4	Comparação de resultados . . . . .	11
3.6.5	Transfer learning . . . . .	12
3.6.5.1	ResNet50 . . . . .	12
3.6.5.2	VGG16 . . . . .	13
3.6.5.3	InceptionV3 . . . . .	13
3.6.5.4	NASNetMobile . . . . .	14
3.6.5.5	Comparação de resultados . . . . .	15
<b>4</b>	<b>Genetic Algorithm</b>	<b>16</b>
4.1	Algoritmo criado . . . . .	16
4.2	Análise de resultados . . . . .	20
<b>Conclusão</b>		<b>22</b>
<b>A</b>	<b>Output completo do 1º algoritmo genético</b>	<b>24</b>
<b>B</b>	<b>Output completo do 2º algoritmo genético</b>	<b>43</b>

# Capítulo 1

## Introdução e Contextualização

As CNN são inspiradas no mecanismo ótico de animais, pois os Neurônios corticais individuais de um animal respondem a estímulos apenas numa certa região do seu campo de visão (campos recetivos). Os campos recetivos de diferentes neurônios sobrepõem-se parcialmente, de forma a cobrir todo o campo de visão.[1] Nos últimos anos as *Convolutional Neural Networks* (CNN) têm sido muito usadas para reconhecimento de padrões, tanto em áreas como *machine learning* como visão por computadores. Um dos aspetos importantes numa CNN é que estas foram desenvolvidas para resolver problemas que não são espacialmente dependentes.[2] Por exemplo, quando se pretende saber a que espécie uma ave pertence, não precisamos de dar relevância à sua localização, ou seja, o que se pretende é apenas deteta-los independente da sua posição. Os algoritmos genéticos são normalmente utilizados para obter soluções aproximadas da ótima, sendo assim um algoritmo de otimização e procura de parâmetros inspirados em processos biológicas como a mutação , *crossover* e seleção. Este algoritmo será utilizado para otimizar as camadas de *feature extraction* de uma rede neuronal convolucional.

Dado isto, o presente relatório será composto por um capítulo relacionado com os *dataset* e as operações sobre o mesmo, um capítulo sobre as redes neurais convolucionais, onde é apresentado a explicação das mesmas e várias possíveis implementações, um capítulo sobre os algoritmos genéticos similar ao anterior, e por último a conclusão, onde é feita uma retrospectiva de todo o trabalho concebido.

Uma nota a salientar é que devido a não ter uma boa GPU recorri à utilização de três contas *google* na tentativa de realizar o máximo de testes possíveis ao modelos criados,no entanto tive várias execuções que falharam devido a falhas de memória (muito recorrentes) , bloqueios de deteção de inatividade, bloqueios de utilização da GPU por parte do colab(devido a grandes períodos de utilização da mesma ), etc. Por tudo isto, tive de fixar alguns parâmetros, tanto nas rede convolucionais como no algoritmo genético, nomeadamente o número de epochs , número de gerações, o número/tamanho de filtros nas camadas convolucionais e nas camadas densas etc.

# Capítulo 2

## Dataset

### 2.1 Descrição do Dataset

O dataset estudado é constituído por imagens de várias espécies de pássaros, sendo que existem 250 espécies diferentes. Os modelos serão treinados com 35215 imagens, sendo que estas não estão distribuídas uniformemente entre as categorias possíveis, porém existem pelo menos 100 imagens de cada espécie. Para validação dos modelos irá ser usado um dataset balanceado, contendo 5 imagens de cada espécie, perfazendo um total de 1250 imagens. O dados de teste são similares ao dados de validação visto que contém exatamente as mesmas características. Todas as imagens são RGB tendo assim três canais, contendo também uma largura e altura de 224 pixels. Uma característica interessante é em relação ao género das espécies, uma vez que, 80% do total das imagens são machos e apenas 20 % são fêmeas, podendo apresentar uma menor *performance* na capacidade de previsão de aves fêmeas. Apresento de seguida uma amostra do dataset:



Figura 2.1: *Overview* do dataset

### 2.2 Data Augmentation

Data Augmentation é uma técnica usada para aumentar os dados fornecidos, adicionando algumas modificações nos dados sendo normalmente algumas transformações geométricas, tais como rotação de uma imagem, espelhar a imagem, colocação de brilho, normalização das matrizes input , distorção etc. Algumas destas técnicas ajudam a reduzir o overfitting de um modelo.[6] Essencialmente, neste dataset foram testadas várias transformações, variando sempre os parâmetros que estas permitem modificar.

- *horizontal flip* - esta operação consiste num *horizontal flip random*, resultando em melhorias imediatas nos modelos. É facilmente perceptível porque os pássaros podem ser fotografados de várias perspectivas. Já o vertical flip houve melhorias ,o que também é compreensível.
- *rotation range* - com esta modificação aumenta-se drasticamente o tempo de execução de cada epoch num model, praticamente 3 vezes mais lento, porém também existiu uma subida de *accuracy* no modelo. Dado isto, para o algoritmo genético esta operação teve de ser removida.
- *brightness range* - existiram algumas mudanças pouco significativas em alguns modelos, e num deles piorou. Por isso decidi remover esta transformação.

- *rescale* - Dado que as redes neuronais funcionam melhor com valores entre 0 e 1 decidi a normalização dos valores das matrizes(imagens).
- *shear range* - Consiste numa distorção ao longo dos 2 eixos da imagem 2D (x,y).Foram obtidos melhores resultados com uma percentagem de 0.1 .
- *zoom* - Realização de zoom na imagem, o que piorou substancialmente o resultado de algumas epochs, dependendo do alcance passado como parametro.Caso esse parametro fosse baixo não obtive melhorias significativas, sendo que no fim .

Estas transformações foram aplicadas com a classe *ImageDataGenerator* do *keras*.

## 2.3 Keras pre-processing

Os modelos de CNN podem ter camadas de pré processamento, tirando assim vantagem da GPU, e não de CPU como é feito no processamento acima,tornando o modelo mais rápido.

A vantagem deste método é na diferença entre o desempenho durante o treino e o desempenho durante a previsão , ou seja, quando se está a fazer uma previsão usando o modelo treinado, esta pode passar por esse pré processamento, o que não acontece com o modelo anterior. Isto pode trazer melhores resultados,sendo que também existem mais formas de pré processamento com estas camadas, nomeadamente o contraste que trouxe uma melhoria significativa aos modelos que serão apresentados.

Dado isto, foram usados os 2 pré processamentos durante a realização deste trabalho.

## 2.4 Batch size

O *batch size* é o número de amostras que irão ser propagadas pela rede em cada iteração numa *epoch*, ou seja, se existirem 1024 imagens no *dataset* para treino e o *batch size* for de 256,a rede irá fazer a primeira iteração/propagação da imagem 1 à 256, sendo a segunda iteração a partir da imagem 257 até 512, e assim sucessivamente.Após cada propagação a rede irá atualizar os pesos. Um valor baixo de *batch* tem a vantagem de usar menos memória e as redes requerem menos tempo de treino, porém valores baixos podem diminuir a estabilidade da rede.

Neste projeto foram usados valores iguais a 32 ou 64 para o *batch size*.

# Capítulo 3

## Convolutional neural network

As *Convolutional Neural Networks* é um tipo de rede neuronal robusta e muito usada para reconhecimento e pré processamento de imagens, que neste caso serão usadas para classificar passáros.

Estas redes podem-se dividir em duas partes, sendo a primeira a extração de características da imagem, tornando assim a segunda num processo de classificação. Tanto na extração de características como o processo de classificação consistem numa composição de camadas que desempenham uma certa função na rede. Praticamente, os *feature maps* recebem uma camada de entrada (dados brutos) e são constituídos por essencialmente em 2 camadas: as camadas convolucionais, que realizam o produto escalar entre o *patch* e os neurónios de uma camada e as camadas de *pool* que tornam a saída das camadas convolucionais mais eficientes a nível da memória computacional. No caso do processo de classificação estes são compostos por uma camada *Fully connected*, ou seja todos os neurónios estão ligados entre si e produzem a classificação final de uma imagem.

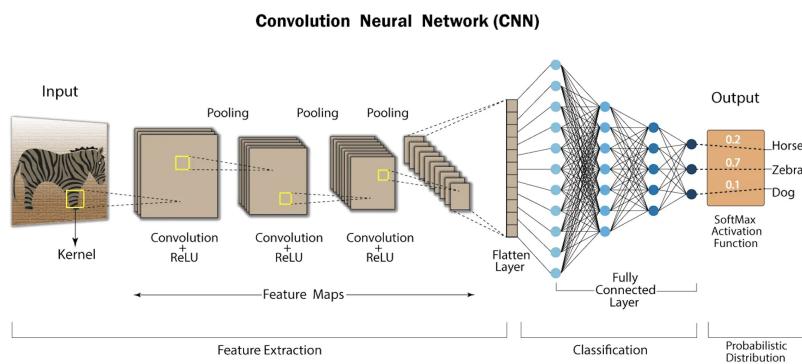


Figura 3.1: *Overview* da Arquitetura de uma CNN

É apresentado de seguida a explicação mais detalhada de cada camada da rede.

### 3.1 Covolution2D Layer

As camadas convolucionais são uma das partes centrais de uma CNN, pois é nestas camadas que se realiza a deteção de features.[5] As camadas iniciais detetam as características de baixo nível, como cantos ou texturas, sendo que as ultimas camadas são usadas para identificar formas e objetos relevantes.[3]

#### 3.1.1 Filters

Numa camada convolucional, os filtros são o número de neurónios dessa camada. Um mapa de características é o resultado da aplicação de um filtro.

### 3.1.2 Kernel

O kernel ( $\mathbf{K}$ ) é representado por uma matriz de pesos que “desliza” sobre os dados de entrada da camada convolucional, sendo executada uma multiplicação de célula a célula entre os inputs da imagem e os pesos do kernel. O resultado desta dá origem ao input da próxima camada convolucional. O resultado desta operação irá produzir uma **diminuição** da matriz de input que poderá ser considerável dependendo do tamanho do kernel.

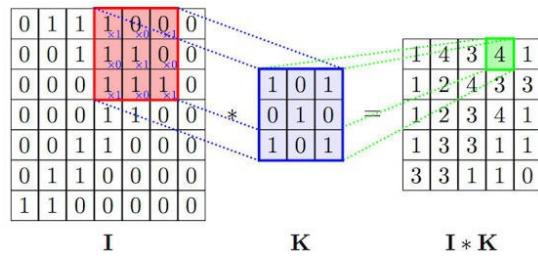


Figura 3.2: Operação realizada pela *convolutional layer*

### 3.1.3 Padding

O *padding* é uma forma de resolver o problema introduzido anteriormente, uma vez que, ao aplicar o processo do *kernel* tendemos a perder pixels no perímetro de nossa imagem. A solução para este problema é adicionar *pixels* extras de preenchimento ao redor do limite de nossa imagem de entrada, aumentando assim o tamanho da imagem. No *tensorflow* existem duas possibilidades para este parâmetro, sendo estas *valid* e *same*. No caso de *valid* não é adicionada nenhuma linha ou colunas de zeros, podendo perder assim alguns pixeis da imagem, como já foi referido anteriormente. No entanto, no caso do *same* percorremos todos os pixeis da imagem, pois é adicionada uma linha ou coluna sempre que for necessário, como se pode ver na seguinte imagem:

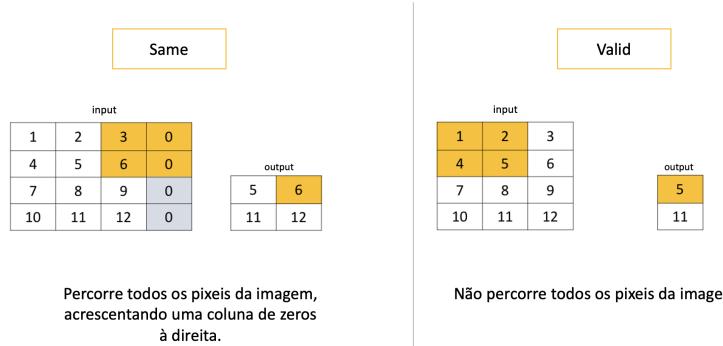


Figura 3.3: Processo de *padding*

### 3.1.4 Activation Function

A função de ativação de um neurónio define o valor de saída deste, dada uma ou mais entradas. O objectivo desta função é introduzir a não linearidade na saída de um neurônio, sendo assim funções não lineares e /ou *down-sampling*.

### 3.1.5 Limitações de uma conv2D

A maior limitação desta layer é o tempo de execução, visto que uma grande quantidade de filtros aumenta drasticamente esse tempo, devido ao aumento da quantidade de cálculos adicionais. Uma das soluções passa por diminuir o tamanho dos filtros e aumentar as strides no maxpooling, reduzindo o campo receptivo de cada filtro e a quantidade de informação que este captura.

## 3.2 Pooling Layer

A operação de *pooling* consiste numa janela fixa que é deslizada sobre todas as regiões de input de forma a reduzir-lo. O *pooling* pode ser efetuado de três formas, sendo estas *Max Pooling*, *Min Pooling* e *Avg Pooling*. Por exemplo, no *Max Pooling* é uma operação de *pooling* que consiste em calcular o valor máximo da janela de um feature map. Similarmente, nos *Min Pooling*, obtém-se o valor mínimo da janela e no *Avg Pooling* faz-se uma média dos valores. Normalmente, esta *layer* é usada após uma camada convolucional.

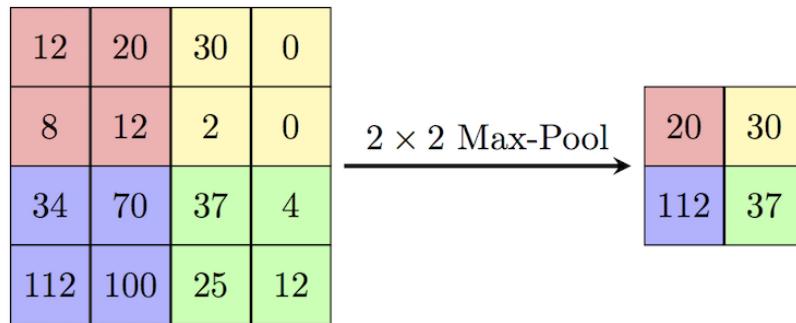


Figura 3.4: Operação de *Max Pooling*

## 3.3 Dropout Layer

As camadas de *dropout* previnem as redes neurais de convergirem para os dados de treino (*Overfitting*), ou seja, serem treinadas em demasia apresentando ótimos resultados para os dados de treino, porém apresentam uma baixa performance na capacidade de previsão em novos dados.

É uma técnica de regularização da rede neuronal que consiste em seleccionar alguns neurónios aleatoriamente sendo estes ignorados numa fase do treino. Isto significa que, a contribuição destes para a função de activação é temporariamente removida, sendo que, os pesos destes também não são modificados na *back-propagation*.[4]

## 3.4 Fully Connected Layer

A última secção da CNN são simplesmente camadas totalmente conectadas também conhecidas como *hidden layers*. Cada neurónio desta layer recebe a entrada de todos os neurónios na camada anterior executando posteriormente a função de activação na camada atual gerando output. Esta camada é representada por um conjunto de *Dense layers*, como se pode ver na seguinte imagem:

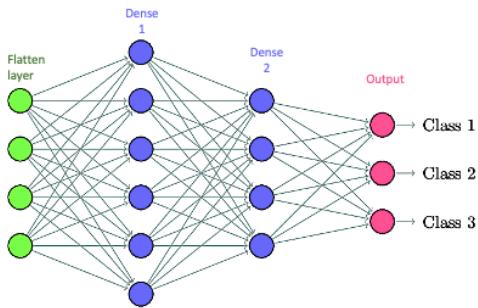


Figura 3.5: Representação de uma Fully Connected Layer

### 3.5 Callbacks

As callbacks são formas que possibilitam a realização de ações durante o treino de um modelo(*fit*). As callbacks utilizadas foram as seguintes:

- **EarlyStopping** - Permite **parar** o treino com base na loss ou accuracy que o modelo está a obter ao longo das *epochs*. Por exemplo, se o modelo já convergiu e ainda faltarem 10 epochs, num cenário normal teríamos de esperar essas 10 epochs. Com este método, conseguimos parar o modelo com base num fator de paciencia. Esse parâmetro permite especificar quantas epochs se quer permitir a convergência. Este foi usado tanto com a loss como a accuracy no dataset de validação.
- **ModelCheckpoint** - permite **guardar** a melhor epoch do processo de *fit* num ficheiro, para posteriormente dar *load* deste, e poder usar essa "CNN" na previsão.
- **ReduceLROnPlateau** - Este operação proporciona uma **redução** do learning rate ao longo do treino. Aqui também existe o fator paciência que tem a mesma funcionalidade. Por exemplo, se for igual a 2, a partir da 3 *epoch* caso a *loss* ou *accuracy* não desça o learning rate irá diminuir uma certa percentagem, passada como parâmetro.

### 3.6 Modelos Criados

#### 3.6.1 Modelo Custom 1

O primeiro modelo criado foi baseado no uso das layers mais comuns, tendo tido analisado algumas CNN já implementadas. Praticamente, o número de neurónios é um valor exponencial com base 2 ( $2^x$ ), começando por  $x=5$  para a primeira camada, sendo posteriormente incrementado 1 unidade ao  $x$  por camada. De seguida, é feito um Maxpooling de com uma matriz (2,2), repetindo este processo 4 vezes. Antes de entrar na *fully connected layer* é feito o flatten. Por último, são adicionadas duas as camadas densas com 512 filtros, alternadas com um *dropout*. Este modelo usa a API *Sequential*, devido a só receber um input.

```

    □ Model: "sequential"
    ┌─────────┐ ┌─────────────────┐ ┌─────────┐
    │ Layer (type)   │ ┌─────────┐ ┌─────────┐
    └─────────┘   │   Output Shape   └ Param # ──
    ┌─────────┐   ┌─────────┐ ┌─────────┐
    conv2d_58 (Conv2D)   (None, 224, 224, 32)   896
    └─────────┘
    max_pooling2d_38 (MaxPooling) (None, 112, 112, 32)   0
    ┌─────────┐
    conv2d_59 (Conv2D)   (None, 112, 112, 64)   18496
    └─────────┘
    max_pooling2d_39 (MaxPooling) (None, 56, 56, 64)   0
    ┌─────────┐
    conv2d_60 (Conv2D)   (None, 56, 56, 128)   73856
    └─────────┘
    max_pooling2d_40 (MaxPooling) (None, 28, 28, 128)   0
    ┌─────────┐
    conv2d_61 (Conv2D)   (None, 28, 28, 256)   295168
    └─────────┘
    max_pooling2d_41 (MaxPooling) (None, 14, 14, 256)   0
    ┌─────────┐
    flatten_14 (Flatten) (None, 50176)   0
    ┌─────────┐
    dense_22 (Dense)   (None, 512)   25690624
    ┌─────────┐
    dropout (Dropout) (None, 512)   0
    ┌─────────┐
    dense_23 (Dense)   (None, 512)   262656
    ┌─────────┐
    dropout_1 (Dropout) (None, 512)   0
    ┌─────────┐
    dense_24 (Dense)   (None, 250)   128250
    └─────────┘
    Total params: 26,469,946
    Trainable params: 26,469,946
    Non-trainable params: 0

```

Figura 3.6: Sumário do modelo 1

Os resultados deste modelo foram os seguintes:

```

result = model.evaluate(test_ds)

40/40 [=====] - 2s 56ms/step - loss: 1.1050 - accuracy: 0.7456

```

Figura 3.7: Resultado do modelo *Sequential* 1

### 3.6.2 Modelo Custom 2

Este segundo modelo foi obtido através de várias iterações de um *Random Search* do *kerastuner*, em que em cada iteração fixava parâmetros diferentes, de forma a, realizar a execução desta procura mais rápida. Assim obtive vários resultados como estão apresentados na figura 3.8, sendo que posteriormente a essa procura, atualizava-se os parâmetros da CNN :

```

... Trial 11 Complete [00h 05m 11s]
val_accuracy: 0.004000000189989805

Best val_accuracy So Far: 0.7088000178337097
Total elapsed time: 00h 58m 35s

Search: Running Trial #12

Hyperparameter | Value          | Best Value So Far
Conv_5_filter  | 416            | 400
Dense_1_units  | 768            | 528
learning_rate   | 0.01           | 0.001
Dense_2_units  | 288            | None

Epoch 1/3

```

Figura 3.8: *Random Search* do *kerastuner*

No fim de algumas iterações resultou o seguinte modelo:

```

    □ Model: "sequential"
    ┌─────────────────────────────────────────────────────────────────┐
    │ Layer (type)          Output Shape       Param #   │
    ├─────────────────────────────────────────────────────────┤
    conv2d (Conv2D)        (None, 224, 224, 32)   896      │
    max_pooling2d (MaxPooling2D) (None, 112, 112, 32) 0      │
    conv2d_1 (Conv2D)       (None, 112, 112, 64)  18496     │
    max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 64) 0      │
    conv2d_2 (Conv2D)       (None, 56, 56, 128) 73856      │
    max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 128) 0      │
    conv2d_3 (Conv2D)       (None, 28, 28, 256) 295168     │
    max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 256) 0      │
    conv2d_4 (Conv2D)       (None, 14, 14, 400) 922000     │
    max_pooling2d_4 (MaxPooling2D) (None, 7, 7, 400) 0      │
    flatten (Flatten)      (None, 19600)        0      │
    dense (Dense)          (None, 528)         10349328    │
    dropout (Dropout)      (None, 528)         0      │
    dense_1 (Dense)        (None, 250)         132250     └─────────────────────────────────────────────────┘
    Total params: 11,791,994
    Trainable params: 11,791,994
    Non-trainable params: 0

```

Figura 3.9: Sumário do modelo 2

Os melhores resultados deste modelo na fase de treino foram os seguintes:

```

551/551 [=====] - 77s 138ms/step - loss: 1.1555 - accuracy: 0.6950 - val_loss: 0.9858 - val_accuracy: 0.7480
Epoch 12/30
551/551 [=====] - 76s 137ms/step - loss: 1.0725 - accuracy: 0.7170 - val_loss: 0.8186 - val_accuracy: 0.7904
Epoch 13/30
551/551 [=====] - 76s 137ms/step - loss: 0.9900 - accuracy: 0.7373 - val_loss: 0.8148 - val_accuracy: 0.7888
Epoch 14/30
551/551 [=====] - 77s 139ms/step - loss: 0.9272 - accuracy: 0.7502 - val_loss: 0.8095 - val_accuracy: 0.7944
Epoch 15/30
551/551 [=====] - 77s 139ms/step - loss: 0.8689 - accuracy: 0.7662 - val_loss: 0.8783 - val_accuracy: 0.7928
Epoch 16/30
551/551 [=====] - 77s 138ms/step - loss: 0.8492 - accuracy: 0.7681 - val_loss: 0.7948 - val_accuracy: 0.8096
Epoch 17/30
551/551 [=====] - 76s 138ms/step - loss: 0.7762 - accuracy: 0.7872 - val_loss: 0.7617 - val_accuracy: 0.8168
Epoch 18/30
551/551 [=====] - 76s 138ms/step - loss: 0.7475 - accuracy: 0.7953 - val_loss: 0.7489 - val_accuracy: 0.8280
Epoch 19/30
551/551 [=====] - 76s 138ms/step - loss: 0.6964 - accuracy: 0.8066 - val_loss: 0.8139 - val_accuracy: 0.8160
Epoch 20/30
551/551 [=====] - 76s 138ms/step - loss: 0.6865 - accuracy: 0.8076 - val_loss: 0.7966 - val_accuracy: 0.8128
Epoch 21/30
551/551 [=====] - 76s 138ms/step - loss: 0.6511 - accuracy: 0.8189 - val_loss: 0.7812 - val_accuracy: 0.8184
Epoch 22/30
551/551 [=====] - 76s 137ms/step - loss: 0.6071 - accuracy: 0.8282 - val_loss: 0.8175 - val_accuracy: 0.8192
Epoch 23/30
551/551 [=====] - 77s 138ms/step - loss: 0.6021 - accuracy: 0.8284 - val_loss: 0.7512 - val_accuracy: 0.8256
Epoch 24/30
551/551 [=====] - 77s 138ms/step - loss: 0.5742 - accuracy: 0.8385 - val_loss: 0.8298 - val_accuracy: 0.8224
Epoch 25/30
551/551 [=====] - 77s 139ms/step - loss: 0.5590 - accuracy: 0.8398 - val_loss: 0.8169 - val_accuracy: 0.8296
Epoch 26/30
551/551 [=====] - 76s 137ms/step - loss: 0.5302 - accuracy: 0.8483 - val_loss: 0.8387 - val_accuracy: 0.8160
Epoch 27/30
551/551 [=====] - 76s 138ms/step - loss: 0.5260 - accuracy: 0.8480 - val_loss: 0.8302 - val_accuracy: 0.8248
Epoch 28/30
551/551 [=====] - 76s 137ms/step - loss: 0.5141 - accuracy: 0.8515 - val_loss: 0.8131 - val_accuracy: 0.8296
Epoch 29/30
551/551 [=====] - 76s 137ms/step - loss: 0.4854 - accuracy: 0.8588 - val_loss: 0.7738 - val_accuracy: 0.8304
Epoch 30/30
551/551 [=====] - 76s 138ms/step - loss: 0.4749 - accuracy: 0.8585 - val_loss: 0.8321 - val_accuracy: 0.8240

```

Figura 3.10: Resultado no treino do modelo *Sequential 2*

É de realçar que este treino foi realizado com 30 *epochs* e de todos os modelos *custom* feitos foi o que obteve os melhores resultados(*customs*).

A melhor *epoch* que se obteve foi a 18, devido a ter uma *accuracy* (validação) a rondar o 83% e obteve a *loss* mais baixa. É um erro comum apenas considerar a *accuracy*, pois nesse caso iríamos para a *epoch* 29, onde a *loss* já era maior, o que poderia ser uma má escolha.

No entanto, como a análise dos outros foi feita com 10 epochs, esta avaliação é com esses valores. A

performance foi a seguinte:

```
results = model2.evaluate(test_ds,return_dict=True,verbose=1)

40/40 [=====] - 2s 51ms/step - loss: 0.9337 - accuracy: 0.7472
```

Figura 3.11: Resultado do modelo *Sequential 2*

Aparentemente, com 10 *epochs* não se nota muita diferença, no entanto, se colocássemos mais *epochs* seria o melhor modelo *custom*.

### 3.6.3 Modelo Custom 3

Este ultimo modelo custom é resultado da aplicação da API *functional* a uma CNN, que permite ter vários inputs em simultâneo. Apresento de seguida o modelo criado:

Model: "model"		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 224, 224, 3]	0
conv2d_5 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_5 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_6 (Conv2D)	(None, 109, 109, 64)	18496
max_pooling2d_6 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_7 (Conv2D)	(None, 52, 52, 128)	73856
max_pooling2d_7 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_8 (Conv2D)	(None, 24, 24, 256)	295168
max_pooling2d_8 (MaxPooling2D)	(None, 12, 12, 256)	0
flatten_1 (Flatten)	(None, 36864)	0
dense_2 (Dense)	(None, 512)	18874880
dropout_1 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 250)	128250
<hr/>		
Total params: 19,391,546		
Trainable params: 19,391,546		
Non-trainable params: 0		

Figura 3.12: Sumário do modelo 3

Os resultados obtidos são similares às redes sequenciais, porém estas redes não foram exploradas ao limite e se fossem, provavelmente obteria-se resultados superiores aos aqui mostrados. Dar inputs diferentes, com processamentos diferentes poderia alterar o cenário. No entanto, como esse não era objectivo principal deste projeto não foi realizado. Assim, com base nos resultados obtidos, este modelo não apresentou qualquer vantagem em relação aos anteriores, como podemos verificar no resultados:

```
result = model3.evaluate(test_ds)

40/40 [=====] - 2s 49ms/step - loss: 1.0378 - accuracy: 0.7472
```

Figura 3.13: Resultado do modelo *Functional 3*

### 3.6.4 Comparação de resultados

De forma a, realizar uma comparação na fase de *fitness* aos modelos é apresentado o seguinte gráfico:

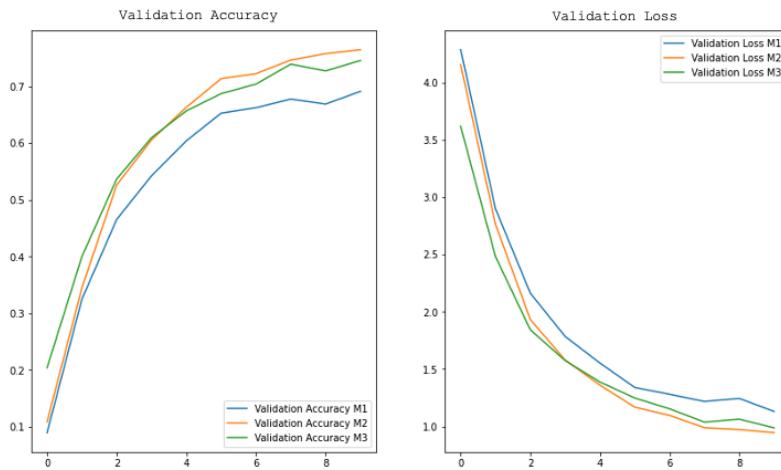


Figura 3.14: Comparação dos resultados de treino dos modelos Custom

Aqui é possível verificar que o melhor modelo tanto a nível de *accuracy* como de *loss* na *validation* (dataset parecido com os dados teste) é o Modelo 2, pois nunca apresenta um fase descendente na *accuracy* nem ascendente na *loss*, o que indica que num teste com mais *epochs* este provavelmente irá continuar a frente dos restantes. Todavia, é importante realçar que os outros 2 modelos também acabam numa fase crescente na *accuracy* e decrescente na *loss*.

### 3.6.5 Transfer learning

#### 3.6.5.1 ResNet50

A *ResNet50* é uma rede com 50 camadas de profundidade, sendo que foi treinada com um *dataset* conhecido por *ImageNet*, ao qual é composto por mais de **14 milhões** de imagens pertencentes a 1000 classes. Esta foi colocada num modelo com três camadas densas de 528 filtros cada uma, sendo que ainda tinham um dropout de 10% entre elas. Apresento de seguida o *summary*:

<code>flatten_2 (Flatten)</code>	<code>(None, 100352)</code>	0	<code>conv5_block3_out[0][0]</code>
<code>fc1 (Dense)</code>	<code>(None, 528)</code>	52986384	<code>flatten_2[0][0]</code>
<code>dropout_2 (Dropout)</code>	<code>(None, 528)</code>	0	<code>fc1[0][0]</code>
<code>fc2 (Dense)</code>	<code>(None, 528)</code>	279312	<code>dropout_2[0][0]</code>
<code>dropout_3 (Dropout)</code>	<code>(None, 528)</code>	0	<code>fc2[0][0]</code>
<code>fc3 (Dense)</code>	<code>(None, 528)</code>	279312	<code>dropout_3[0][0]</code>
<code>predicts (Dense)</code>	<code>(None, 250)</code>	132250	<code>fc3[0][0]</code>
<hr/>			
Total params: 77,264,970			
Trainable params: 53,677,258			
Non-trainable params: 23,587,712			

Figura 3.15: Sumário do modelo com *ResNet50*

Este Modelo teve a seguinte performance na previsão nos dados de teste:

```

results =resnet_model.evaluate(test_ds,return_dict=True,verbose=1)

40/40 [=====] - 15s 366ms/step - loss: 0.1231 - accuracy: 0.9672

```

Figura 3.16: Resultado do modelo com *ResNet50*

### 3.6.5.2 VGG16

A VGG16 é uma arquitetura de uma CNN criada por um grupo na universidade de Oxford. Esta também é um modelo pré treinado com 23 camadas de profundidade, sendo que foi treinada com o mesmo *dataset* referido anteriormente (*ImageNet*). A estrutura de camadas Densas também foi a mesma, de forma a, poder realizar uma comparação na seção 3.6.5.4. Apresento de seguida o *summary*:

<code>flatten (Flatten)</code>	(None, 25088)	0
<code>fc1 (Dense)</code>	(None, 528)	13246992
<code>dropout (Dropout)</code>	(None, 528)	0
<code>fc2 (Dense)</code>	(None, 528)	279312
<code>dropout_1 (Dropout)</code>	(None, 528)	0
<code>fc3 (Dense)</code>	(None, 528)	279312
<code>predicts (Dense)</code>	(None, 250)	132250
<hr/>		
Total params: 28,652,554		
Trainable params: 13,937,866		
Non-trainable params: 14,714,688		

Figura 3.17: Sumário do modelo com *VGG16*

Este Modelo teve a seguinte performance na previsão nos dados de teste:

```

results=vgg16_model.evaluate(test_ds,return_dict=True,verbose=1)

40/40 [=====] - 14s 349ms/step - loss: 0.4556 - accuracy: 0.8784

```

Figura 3.18: Resultado do modelo com *VGG16*

É possível verificar que em 10 epochs o modelo *ResNet50* apresentou melhores resultados na previsão, no entanto não garante que no futuro este não possa ter melhores resultado, visto que pode ter um *learning rate* mais lento. É importante realçar que tanto o número de camadas densas como o número de filtros pode não ser o ótimo para este tipo de rede.

### 3.6.5.3 InceptionV3

A InceptionV3 é uma rede que foi criada pela *Google*, sendo composta por 159 camadas de profundidade que, mais uma vez foi treinada com o dataset *ImageNet*. Assim, este modelo apresenta a seguinte arquitetura:

<code>flatten (Flatten)</code>	(None, 51200)	0	<code>mixed10[0][0]</code>
<code>fc1 (Dense)</code>	(None, 528)	27034128	<code>flatten[0][0]</code>
<code>dropout (Dropout)</code>	(None, 528)	0	<code>fc1[0][0]</code>
<code>fc2 (Dense)</code>	(None, 528)	279312	<code>dropout[0][0]</code>
<code>dropout_1 (Dropout)</code>	(None, 528)	0	<code>fc2[0][0]</code>
<code>fc3 (Dense)</code>	(None, 528)	279312	<code>dropout_1[0][0]</code>
<code>predicts (Dense)</code>	(None, 250)	132250	<code>fc3[0][0]</code>
<hr/>			
Total params: 49,527,786			
Trainable params: 27,725,002			
Non-trainable params: 21,802,784			

Figura 3.19: Sumário do modelo com *VGG16*

Este Modelo teve a seguinte performance na previsão nos dados de teste:

```
result = incept_v3.evaluate(test_ds)

40/40 [=====] - 13s 320ms/step - loss: 0.3030 - accuracy: 0.9168
```

Figura 3.20: Resultado do modelo com *InceptionV3*

Neste modelo obtemos bons resultados para este número de epochs(10), pois temos uma boa *accuracy* e uma *loss* baixa. Sendo assim, este modelo apresenta-se em segundo lugar, apenas abaixo do *ResNet50*.

### 3.6.5.4 NASNetMobile

As *NASNetMobile* são utilizam uma arquitectura interessante, visto que, possibilitam a separação do *kernel* em dois nas camadas convolucionais. Por exemplo, em vez de existir um *kernel*  $3 \times 3$ , seria um *kernel*  $3 \times 1$  e um *kernel*  $1 \times 3$ . Esta redução reduz o número de operações necessárias para a convolução, tornado assim a rede mais rápida. No entanto nem sempre é possível a separação na dimensão espacial, por isso é mais comum separar na profundidade (*channels*). Apresento de seguida o *summary*:

<code>flatten_9 (Flatten)</code>	(None, 51744)	0	<code>activation_187[0][0]</code>
<code>fc1 (Dense)</code>	(None, 528)	27321360	<code>flatten_9[0][0]</code>
<code>dropout_18 (Dropout)</code>	(None, 528)	0	<code>fc1[0][0]</code>
<code>fc2 (Dense)</code>	(None, 528)	279312	<code>dropout_18[0][0]</code>
<code>dropout_19 (Dropout)</code>	(None, 528)	0	<code>fc2[0][0]</code>
<code>fc3 (Dense)</code>	(None, 528)	279312	<code>dropout_19[0][0]</code>
<code>predicts (Dense)</code>	(None, 250)	132250	<code>fc3[0][0]</code>
<hr/>			
Total params: 32,281,950			
Trainable params: 28,012,234			
Non-trainable params: 4,269,716			

Figura 3.21: Comparação dos resultados de treino dos modelos TL.

Este Modelo teve a seguinte performance na previsão nos dados de teste:

```
40/40 [=====] - 14s 360ms/step - loss: 0.4602 - accuracy: 0.8776
```

Figura 3.22: Resultado do modelo com *NASNetMobile*

Aqui temos o último modelo, onde é possível verificar que é significativamente mais fraco que o *VGG16*, tornado-se assim o pior modelo (com base nestes resultados). No entanto como já foi dito anteriormente, com este número de *epochs* não é possível garantir que este modelo não seja o melhor ou o pior.

### 3.6.5.5 Comparação de resultados

Nesta secção é apresentada uma comparação do treino entre os modelos de *Transfer Learning*, onde vai ser possível fazer uma breve análise da *loss* e *accuracy* com base no *dataset* de validação, que é muito parecido ao de teste.

É de realçar as redes foram todas executadas com a mesma função de otimização, que é a *Adagrad*.

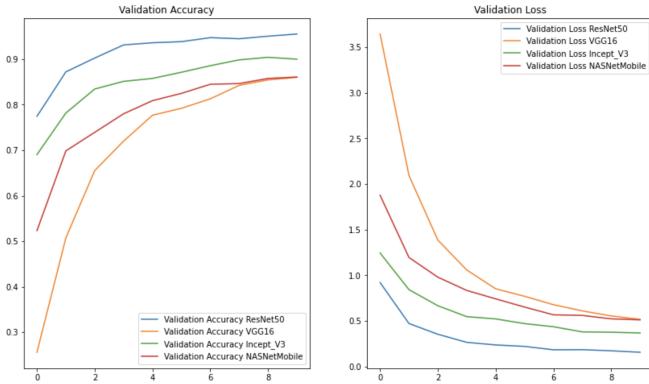


Figura 3.23: Comparação dos resultados de treino dos modelos TL.

Com base neste gráfico é possível verificar que o modelo que obteve melhores resultado foi o *ResNet50*. Este está em todas as *epochs* com uma melhor performance que os outros, o que poderia ser um critérios de escolha, visto que precisa de menos iterações para ter resultados razoáveis. É de realçar que o modelo da *VGG16* apresentou um enorme crescimento criando assim uma perspetiva positiva. Além disso, na última *epoch* de treino conseguiu atingir o menor valor de *loss* da *NASNetMobile*, o que é um bom "pressentimento", dado o seu percurso. Todavia, não deixa de ser um valor pequeno de iterações nas CNN deste calibre, pois estas poderiam facilmente obter melhores resultados com outras configurações.

Outro ponto relevante é a importância da função de **optimização** visto que, esta pode piorar bastante os resultados obtidos, como se pode ver na seguinte imagem, onde foi usada a função *adam*:

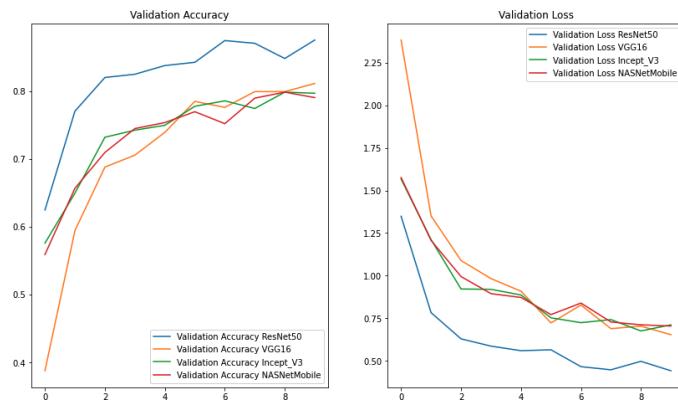


Figura 3.24: Comparação dos resultados de treino dos modelos TL.

As redes foram novamente submetidas ao treino com a função de otimização diferente (*adam*), mantendo as mesmas configurações previamente apresentadas. É possível verificar que a *ResNet50* mantém-se na frente, porém o valor das *accuracies* baixo em todos os modelos, mostrando assim a importância da função de otimização.

# Capítulo 4

## Genetic Algorithm

O algoritmo genético ou *genetic algorithm* (GA) é um algoritmo de **optimização** inspirado na biologia com base no cruzamento dos melhores indivíduos da população, tendo como objectivo criar indivíduos mais fortes. Estes, normalmente são conhecidos como os *parents*, sendo que os restantes são os *childs*. Aos indivíduos mais fracos habitualmente são aplicadas mudanças tendo em conta algumas características dos *parents*, sendo também realizada uma mutação aleatória em alguns genes para a tentativa de melhorar a aptidão dos mesmos.

Existem vários conceitos importantes para realçar:

- **Cromossoma** - representa um individuo no algoritmo genérico;
- **Gene** - Características de um indivíduo;
- **População** - representa um conjunto de indivíduos;
- **Seleção** - Escolha dos melhores indivíduos de uma população;
- **Crossover** - Dois dos melhores cromossomas são escolhidos para conceder genes a um individuo mais fraco;
- **Mutação** - Ocorre nos cromossomas filhos (*childs*) e consiste na alteração de alguns genes aleatórios, na tentativa de melhorar a sua aptidão;
- **Fitness** - Função responsável por avaliar um cromossoma;

### 4.1 Algoritmo criado

Com base nos resultados obtidos, dado que os modelos com *transfer learning* apresentaram resultados satisfatórios, visto que um deles obteve 97% *accuracy* no dataset teste, foi optado por realizar a optimização em modelos *custom*, visto que a mistura dos dois métodos não iria ser útil devido à escolha tender sempre para esse tipo de modelos pré treinados.

O algoritmo genético criado tem como objetivo otimizar as camadas que representam a *feature extraction* de uma CNN, uma vez que, é aqui que existem mais parâmetros a otimizar. Posteriormente também foram feitas umas pequenas mudanças nas camadas Densas para ver a variabilidade dos resultados obtidos. Para a criação deste algoritmo, foi seguida a seguinte estratégia:

#### 1. Definir cromossoma

Para começar a implementar um GA é necessário uma ter uma estruturação devidamente ponderada. Sendo assim primeiro define-se o que seria um cromossoma, que neste caso estamos a falar de uma CNN. Dado que, neste contexto não existe vantagem em usar um modelo funcional, apenas foi implementado CNN sequenciais. Dado isto, foi criada uma *class* CNN que tem várias operações sobre a

mesma, tais como, a adição de vários tipos de camadas, compilação do modelo etc.

## 2. Definir Genes

De seguida, após ter a estrutura de um individuo da população(cromosoma), foram escolhidas as características que se pretende otimizar/ alterar. Como já foi dito anteriormente, o objetivo era otimizar os parâmetros das *layers Covolution2D* e as de *Pooling*. Inicialmente, também foi definido como gene a ordem das camadas de *Pooling* (`genes['nlayers_Pooling']`),porém para *poolings* menores que o número de *layers* a GPU do colab ficava sempre sem memória (`resourceexhaustederror`), o que limitou essa otimização.No entanto, deixei em comentário essa implementação, menos na operação de *crossover*, pois a função apresentava um grande dimensão e de forma a melhorar a legibilidade do código decidi remover. Todavia, essa remodelação , em termos de complexidade, não é assim tão complexa de realizar.

Para implementar isto, foi criado um dicionário que continha todos os parâmetros possíveis iniciais das CNN, de forma a , não escolher valores totalmente *randoms* na inicialização da CNN mas sim posições *random* destas listas.Mostra-se de seguida, um exemplo disso:

```
genes={  
    'nfilters':[32,64,128,180,200,256],#,300,400],  
    'kernel_size':[(3,3)], #,(5,5),(7,7)]  
    'padding':['same'],  
    'activation':['relu'],  
    'pool_size':[(2,2)],  
    'strides':[(2,2)],  
    'nlayers_Pooling': [n_layers]#[x for x in range(n_layers+1) if x > n_layers/3]  
}
```

Figura 4.1: Representação dos Genes no *python*

## 3. Seleção dos Pais

Nesta operação é definida a forma como cada individuo é avaliada com base em heurísticas, tal com a accuracy , a loss etc.No algoritmo criado apenas são selecionadas com base na accuracy no dataset de teste, sendo recebido como parâmetro o número de país que se pretende na população e os seus resultados no *evaluate* de cada CNN.

## 4. Crossover

Após a seleção dos melhores indivíduos explicada anteriormente, é feito o *crossover* com esses pais, criando novos indivíduos para a população. A troca é feita entre camadas convolucionais e camadas de *pooling*,ou seja,o filho irá ter a 1<sup>a</sup> camada *Conv2D* do primeiro pai, sendo de seguida a 2<sup>a</sup> camada *conv2D* do pai 2. O mesmo acontece com as camadas de *pooling*, como se pode verificar na figura 4.2 . Dado as limitações, do *colab* tive de limitar todos os parâmetros excepto os *filters*, pois o tempo necessário para cada *epoch* aumenta facilmente.No entanto, basta aumentar as possibilidades no dicionários dos genes apresentados anteriormente.

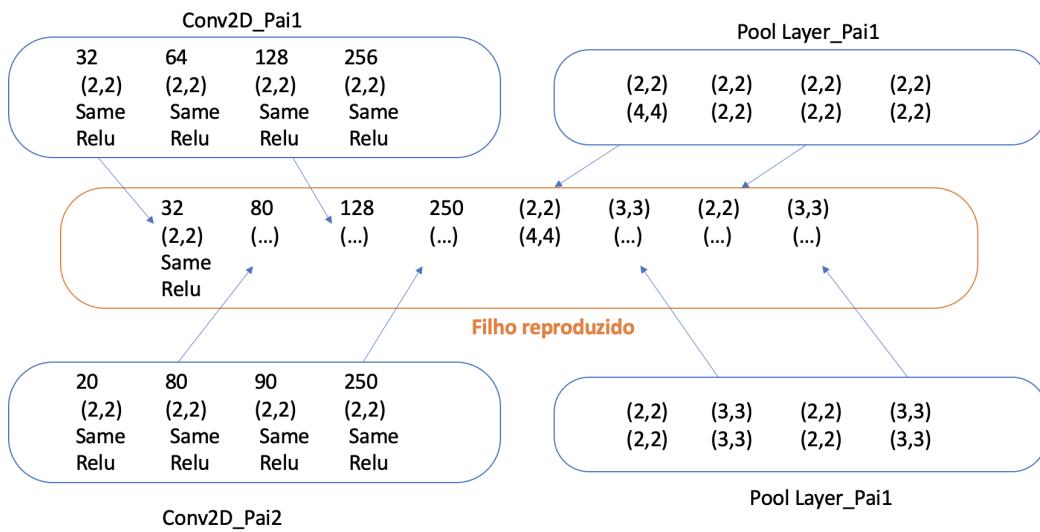


Figura 4.2: Trocas que ocorrem no *Crossover*

## 5. Mutação

Nesta operação para não usar os valores pré definidos nos genes e aumentar as probabilidades de existir uma maior variância de valores, conseguindo pretender assim obter uma solução mais próxima da ótima. Outra hipótese seria gerar um valor aleatório entre 32 e 400, porem o problema é que podemos ir de uma camada de 400 filtros para uma 32 ou vice-versa. Para combater estes fenómenos foram usadas 2 tipo de distribuições:

- Distribuição de Gauss (Normal)

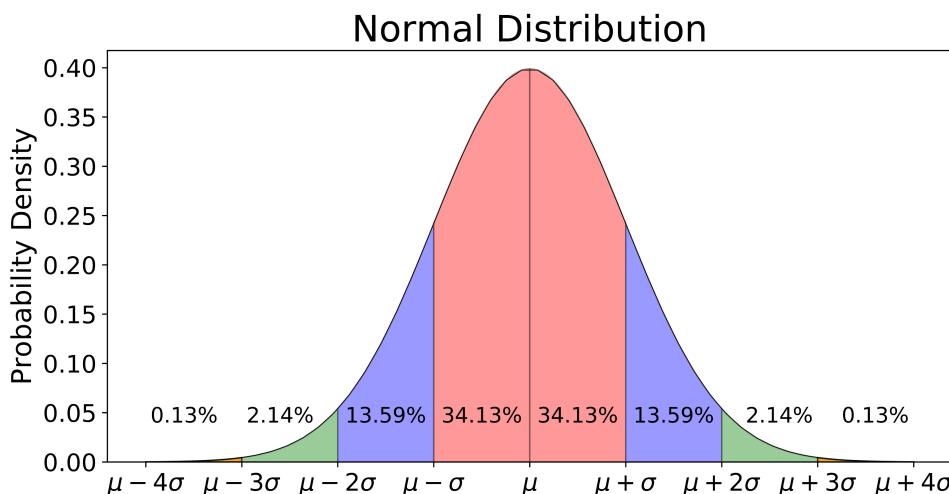


Figura 4.3: Distribuição Normal

Após o *crossover*, esse filho reproduzido é submetido ao processo de mutação, que consiste em selecionar os filtro reproduzidos, e atribuir esse valor como média da distribuição, sendo que o desvio padrão é metade desse valor, calculando assim  $n$  valores da distribuição (o  $n$  utilizado foi 2). Após esse processo é feito a média desses valores, para tornar mais provável um mutação entre  $\mu + \sigma$  e os  $\mu - \sigma$ , como se pode ver na figura 4.3. Estes parâmetros podem todos ser facilmente alterados.

- Distribuição Beta

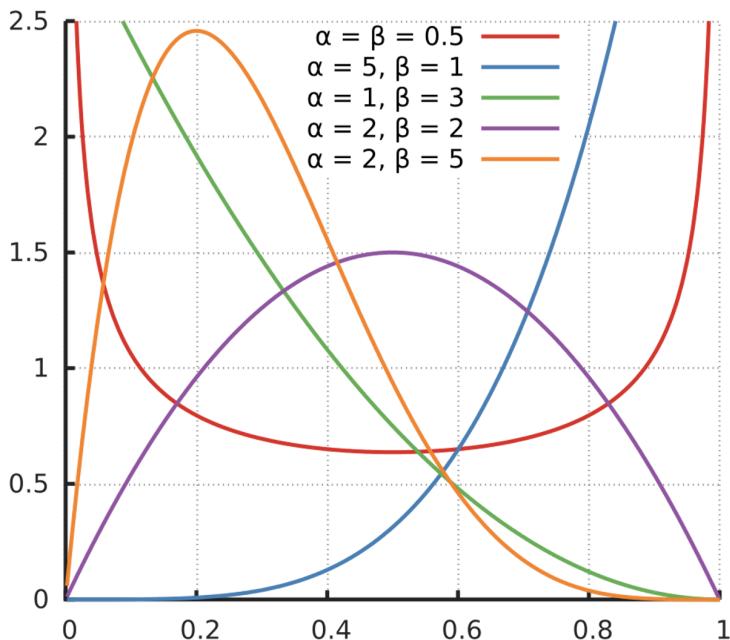


Figura 4.4: Distribuição Normal

Com base numa pesquisa feita por *Guillaume, Martin e Thomas Lenormand* [7], foi concluído que uma das melhores distribuições para aproximar a solução da óptima é a Beta. É fácil de perceber que esta distribuição traz mais aleatoriedade ao sistema, e por isso foi adicionada, pois apresenta vários comportamentos dependendo de  $\alpha$  e  $\beta$ . Para isso, são gerados dois valores entre 1 e 5 sendo atribuídos a  $\alpha$  e  $\beta$ . Esta distribuição dá sempre valores entre 0 e 1, e com este output é necessário a realização de uma função que com base nesse valores aumente ou decrescente o número de filtros. A função criada é a seguinte:

$$F(x, v) = \begin{cases} v + x \times v & \text{if } x > 0.3 \\ v - x \times v & \text{otherwise} \end{cases}$$

sendo  $x$  o calculado pela da distribuição, e  $v$  o valor atual do filtro. Praticamente, é removido ou adicionada a percentagem retornada pela distribuição ao filtro atual arredondando esse valor ao unidades. No entanto, não consegui obter resultados significativos com esta função e devido a isso, na análise apresentada posteriormente, esta não foi utilizada. Devido também à falta de recurso e de tempo não realizei os testes necessários para explorar mais esta distribuição.

## 6. Geração da população Inicial

Para a geração da população inicial, são gerados números aleatórios entre 0 e o tamanho de cada lista no dicionário de genes.Também são geradas logos todas as configurações de todos os indivíduos(CNN) da população. Apresento, de seguida um exemplo de um possível *output* desta operação, sendo que o número de camadas convolucionais está fixado em 4 e o tamanho da população é 6.Na figura abaixo, cada linha de cada matrizes representa a mesma CNN,ou seja, o conjunto de todas as linhas 0 das matrizes constroem a configuração da 1<sup>a</sup> CNN.Também existe um array onde a primeira posição deste é o número de camadas de pooling que essa CNN vai ter. No entanto, como já foi referido anteriormente, não consegui concluir o *fitting* dessas CNN's (geradas com camadas inferiores ao n<sup>o</sup> de *Conv2D*), devido à memoria da GPU do colab não ser suficiente.

```
#           filters      /   kernel size    /   Padding      /   Activation
array([[[[0, 0, 5, 2],      [[0, 0, 0, 0],    [[0, 0, 0, 0],    [[0, 0, 0, 0],
        [3, 5, 5, 4],      [0, 0, 0, 0],    [0, 0, 0, 0],    [0, 0, 0, 0],
        [0, 0, 3, 4],      [0, 0, 0, 0],    [0, 0, 0, 0],    [0, 0, 0, 0],
        [1, 5, 4, 1],      [0, 0, 0, 0],    [0, 0, 0, 0],    [0, 0, 0, 0],
        [5, 3, 4, 1],      [0, 0, 0, 0],    [0, 0, 0, 0],    [0, 0, 0, 0],
        [3, 2, 3, 2]],     [0, 0, 0, 0]],    [0, 0, 0, 0]]    [0, 0, 0, 0]]]),
# Numero de pooling2d para cada CNN
array([4, 4, 4, 4, 4, 4]),
#           Pool size      /   Strides para cada CNN
array([[[[0, 0, 0, 0],      [[0, 0, 0, 0],
        [0, 0, 0, 0],      [0, 0, 0, 0],
        [0, 0, 0, 0],      [0, 0, 0, 0],
        [0, 0, 0, 0],      [0, 0, 0, 0],
        [0, 0, 0, 0],      [0, 0, 0, 0],
        [0, 0, 0, 0]],     [0, 0, 0, 0]]])
```

Figura 4.5: Representação da população inicial na implementação

## 7. Critéiro de Terminação do GA

O algoritmo genético pode terminar por 2 motivos:

- Quando uma CNN atinge um *threshold* passado como parâmetro.
- Quando chegamos ao fim das gerações pré definidas,também passadas como parâmetro.

## 4.2 Análise de resultados

Estes algoritmos genéticos foram executados com as seguintes caraterísticas:

- Todas os modelos nos 3 algoritmos genéticos têm 4 *layers Conv2D*;
- Foram executadas 5 *epoch* por modelo;

- Como critério de paragem, caso atingissem 90% de *accuracy* ou executar as 5 gerações o algoritmo para.
- O tamanho da população é 6 indivíduos (CNN's).

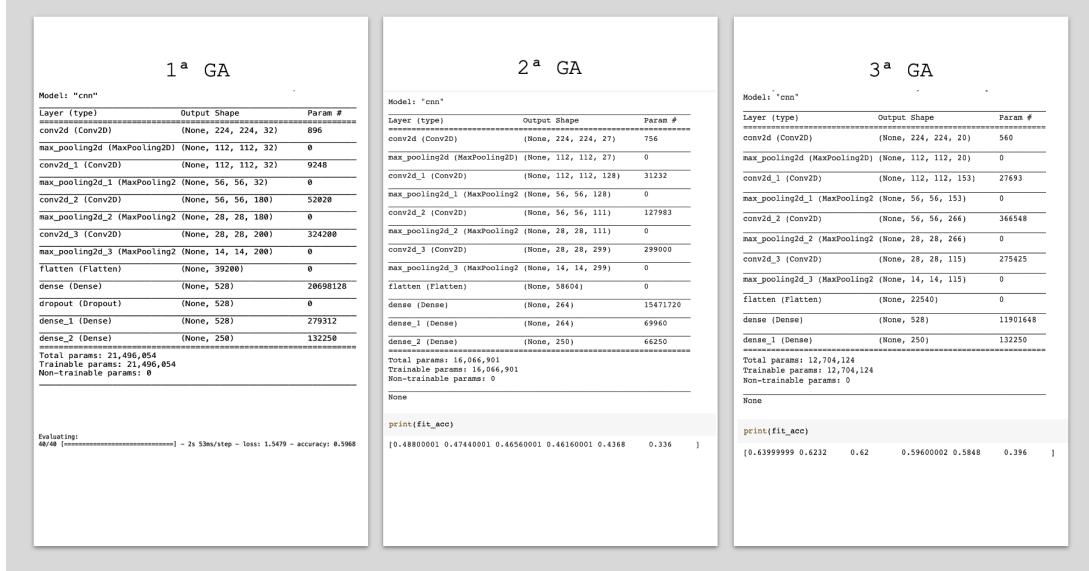


Figura 4.6: Resultados de 3 GA diferentes

Uma vez que, o objetivo deste algoritmo é otimizar os parâmetros colocados nos genes, estes algoritmos genéticos diferem nas camadas *fully connected*, verificando assim que estas têm influência no processo de classificação, como foi dito anteriormente.

É possível verificar que a primeira GA apresentada retornou o melhor modelo com uma *accuracy* próxima dos 60% com uma *loss* de 1.55. Esta contém duas camadas densas de 528 filtros contendo *dropout* de 10% no meio de delas. Este resultados são bons dado que o modelo tem apenas 5 *epochs*.

Na terceira GA é apresentado com resultado final do algoritmo uma rede com apenas temos 1 camada densa de 528 filtros (sem contar com a camada densa final) que obteve os melhores resultados contendo uma *accuracy* de 64%. Na segunda GA o objetivo era verificar se uma camada densa de 528 filtros era equivalente a duas camadas densas de 264, o que não se verificou dado que o melhor modelo obtido teve apenas 48 % de *accuracy* no teste.

Portanto, a GA que apresentou melhor performance foi a terceira, no entanto as características que foram descritas anteriormente não são as melhores (devido a baixo poder computacional pois estes algoritmos já demoram perto de 5 horas a executar no colab e dependendo da *data argumentation* e outros fatores pode escalar facilmente). Outro ponto a salientar, com base nos resultados obtidos, é que o número de filtros nas camadas densas não devem ser **todas** muito próximas ao número de classes existentes(2ª GA).

Foi colocada em apêndice a total execução dos 2 primeiros algoritmos genéticos, visto que, a terceira execução não foi guardada.

# Conclusão

Durante a realização deste projeto, tive a oportunidade de consolidar e desenvolver os conhecimentos abordados nesta unidade curricular como, por exemplo, o desenvolvimento de uma *Convolutional Neural Network*, a utilização de *data augmentation* para melhorar os modelos, a otimização das CNN's, isto é, a utilização de um algoritmo genético para esse propósito, etc.

Os melhores modelos realizados foram os de *transfer learning* contendo ótimos resultados para o número de *epochs* executados, e por isso é que não foram otimizados. Nos modelos *custom* é importante realçar que não foi conseguido uma melhor otimização devido à falta de recursos computacionais, visto que existiram muitos erros por falta de memória na GPU que bloquearam o progresso do projeto, pois uma das melhorias que gostava de ter testado era o número de camadas de *pooling*, perdendo assim menos informação sobre a imagem, podendo resultar em melhores resultados. Este seria um bom tópico como trabalho futuro, com melhores recursos. Outro tópico relevante, que não foi tão explorado neste projeto foi a relevância das camadas de classificação (*fully connected layers*) e a sua ordem tem muita influencia com o crescimento/decrescimento do número de filtros nessas camadas.

No entanto, os principais processos envolvidos tanto nas *Convolutional Neural Network* como nos *genetic algorithms* foram bem explorados neste projeto.

Em suma, considero que o trabalho produzido cumpre todos os requisitos propostos, contento alguns pontos relevantes que não foram explicitamente pedidos, resultando implementação na construção de vários modelos *custom* e modelos pré treinados capazes de prever com uma *accuracy* razoável as várias espécies de pássaros.

# Bibliografia

- [1] MATSUGU, MASAKAZU; KATSUHIKO MORI; YUSUKE MITARI; YUJI KANEDA (2003) '**Subject independent facial expression recognition with robust face detection using a convolutional neural network**' doi:10.1016/S0893-6080(03)00115-1
- [2] SAAD ALBAWI , TAREQ ABED MOHAMMED ,SAAD AL-ZAWI (2017) '**Understanding of a Convolutional Neural Network**' ICET2017, Antalya, Turkey
- [3] ZEILER, M. D. & FERGUS, R. (2014).'**Visualizing and understanding convolutional networks.** In European conference on computer vision'. Springer
- [4] NITISH SRIVASTAVA, GEOFFREY HINTON, ALEX KRIZHEVSKY, ILYA SUTSKEVER, RUSLAN SALAKHUTDINOV(2014) '**Dropout: A Simple Way to Prevent Neural Networks from Overfitting**' Journal of Machine Learning
- [5] F. SULTANA, A. SUFIAN AND P. DUTTA, (2018)'**Advancements in Image Classification using Convolutional Neural Network**' Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)
- [6] SHORTEN, C., KHOSHGOFTAAR, T.M.(2019) '**A survey on Image Data Augmentation for Deep Learning.**' J Big Data 6
- [7] MARTIN, GUILLAUME, AND THOMAS LENORMAND. '**The distribution of beneficial and fixed mutation fitness effects close to an optimum.**' Genetics vol. 179,2 (2008): 907-16. doi:10.1534/genetics.108.087122

## Apêndice A

# Output completo do 1º algoritmo genético

```
(array([[[0, 0, 5, 2],
       [3, 5, 5, 4],
       [0, 0, 3, 4],
       [1, 5, 4, 1],
       [5, 3, 4, 1],
       [3, 2, 3, 2]],

      [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]],

      [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]],

      [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]],

      [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]],

      [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]]],  
      dtype=uint8), array([4, 4, 4, 4, 4, 4]), array([[[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]]],  
      dtype=uint8))  
1º Geration  
Model: "cnn"
```

---

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```

=====
conv2d (Conv2D)           (None, 224, 224, 32)      896
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
-----
conv2d_1 (Conv2D)          (None, 112, 112, 32)     9248
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 32)      0
-----
conv2d_2 (Conv2D)          (None, 56, 56, 256)    73984
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 256)      0
-----
conv2d_3 (Conv2D)          (None, 28, 28, 128)   295040
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 128)      0
-----
flatten (Flatten)          (None, 25088)            0
-----
dense (Dense)              (None, 528)             13246992
-----
dropout (Dropout)          (None, 528)            0
-----
dense_1 (Dense)            (None, 528)            279312
-----
dense_2 (Dense)            (None, 250)             132250
=====

Total params: 14,037,722
Trainable params: 14,037,722
Non-trainable params: 0

Fitting Model 1 ...

Epoch 1/10
551/551 [=====] - 178s 318ms/step - loss: 5.5240 - accuracy: 0.0063 -
val_loss: 5.5298 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 174s 316ms/step - loss: 5.5066 - accuracy: 0.0113 -
val_loss: 5.5355 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5355 - accuracy: 0.0040
Model: "cnn"

=====
Layer (type)           Output Shape        Param #
=====
conv2d (Conv2D)          (None, 224, 224, 180)      5040
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 180)      0
-----
conv2d_1 (Conv2D)          (None, 112, 112, 256)    414976
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)      0
-----
conv2d_2 (Conv2D)          (None, 56, 56, 256)    590080
=====
```

```

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 200) 461000  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 200) 0  

-----  

flatten (Flatten) (None, 39200) 0  

-----  

dense (Dense) (None, 528) 20698128  

-----  

dropout (Dropout) (None, 528) 0  

-----  

dense_1 (Dense) (None, 528) 279312  

-----  

dense_2 (Dense) (None, 250) 132250  

=====  

Total params: 22,580,786  

Trainable params: 22,580,786  

Non-trainable params: 0  

-----  

Fitting Model 2 ...  

Epoch 1/10  

551/551 [=====] - 181s 323ms/step - loss: 5.5319 - accuracy: 0.0066 -  

val_loss: 5.5318 - val_accuracy: 0.0040  

Epoch 2/10  

551/551 [=====] - 178s 322ms/step - loss: 5.5049 - accuracy: 0.0087 -  

val_loss: 5.5311 - val_accuracy: 0.0040  

Evaluating:  

40/40 [=====] - 2s 55ms/step - loss: 5.5311 - accuracy: 0.0040  

Model: "cnn"  

-----  

Layer (type) Output Shape Param #  

=====  

conv2d (Conv2D) (None, 224, 224, 32) 896  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 32) 0  

-----  

conv2d_1 (Conv2D) (None, 112, 112, 32) 9248  

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 180) 52020  

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 180) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 200) 324200  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 200) 0  

-----  

flatten (Flatten) (None, 39200) 0  

-----  

dense (Dense) (None, 528) 20698128  

=====
```

```

dropout (Dropout)           (None, 528)          0
-----
dense_1 (Dense)            (None, 528)          279312
-----
dense_2 (Dense)            (None, 250)          132250
-----
Total params: 21,496,054
Trainable params: 21,496,054
Non-trainable params: 0
-----
Apaguei este resultado sem querer.
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 1.5479 - accuracy: 0.5968

```

Model: "cnn"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_1 (Conv2D)	(None, 112, 112, 256)	147712
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 256)	0
conv2d_2 (Conv2D)	(None, 56, 56, 200)	461000
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 200)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	115264
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 528)	6623760
dropout (Dropout)	(None, 528)	0
dense_1 (Dense)	(None, 528)	279312
dense_2 (Dense)	(None, 250)	132250

```

Total params: 7,761,090
Trainable params: 7,761,090
Non-trainable params: 0
-----
Fitting Model 4 ...

```

```

Epoch 1/10
551/551 [=====] - 178s 320ms/step - loss: 5.4760 - accuracy: 0.0092 -
val_loss: 5.0172 - val_accuracy: 0.0424
Epoch 2/10
551/551 [=====] - 176s 320ms/step - loss: 4.9553 - accuracy: 0.0433 -
val_loss: 4.4183 - val_accuracy: 0.0976
Epoch 3/10
551/551 [=====] - 177s 321ms/step - loss: 4.2972 - accuracy: 0.1139 -
val_loss: 3.6769 - val_accuracy: 0.1944
Epoch 4/10
551/551 [=====] - 176s 320ms/step - loss: 3.6325 - accuracy: 0.2143 -
val_loss: 3.1785 - val_accuracy: 0.2640
Epoch 5/10
551/551 [=====] - 176s 320ms/step - loss: 3.2401 - accuracy: 0.2797 -
val_loss: 2.6646 - val_accuracy: 0.3800
Epoch 6/10
551/551 [=====] - 175s 318ms/step - loss: 2.8012 - accuracy: 0.3692 -
val_loss: 2.3633 - val_accuracy: 0.4376
Epoch 7/10
551/551 [=====] - 175s 318ms/step - loss: 2.4738 - accuracy: 0.4196 -
val_loss: 2.1356 - val_accuracy: 0.4904
Epoch 8/10
551/551 [=====] - 176s 319ms/step - loss: 2.1871 - accuracy: 0.4786 -
val_loss: 1.9438 - val_accuracy: 0.5248
Epoch 9/10
551/551 [=====] - 175s 318ms/step - loss: 1.9465 - accuracy: 0.5254 -
val_loss: 1.8484 - val_accuracy: 0.5432
Epoch 10/10
551/551 [=====] - 176s 319ms/step - loss: 1.7513 - accuracy: 0.5670 -
val_loss: 1.7527 - val_accuracy: 0.5768
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 1.7141 - accuracy: 0.5704
Model: "cnn"

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 256)	7168
max_pooling2d (MaxPooling2D)	(None, 112, 112, 256)	0
conv2d_1 (Conv2D)	(None, 112, 112, 180)	414900
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 180)	0
conv2d_2 (Conv2D)	(None, 56, 56, 200)	324200
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 200)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	115264
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0

```

dense (Dense)           (None, 528)          6623760
-----
dropout (Dropout)       (None, 528)          0
-----
dense_1 (Dense)         (None, 528)          279312
-----
dense_2 (Dense)         (None, 250)          132250
=====
Total params: 7,896,854
Trainable params: 7,896,854
Non-trainable params: 0
-----
Fitting Model 5 ...
Epoch 1/10
551/551 [=====] - 179s 321ms/step - loss: 5.5206 - accuracy: 0.0059 -
val_loss: 5.5343 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 176s 319ms/step - loss: 5.5081 - accuracy: 0.0088 -
val_loss: 5.5333 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 5.5333 - accuracy: 0.0040
Model: "cnn"
-----
Layer (type)            Output Shape        Param #
=====
conv2d (Conv2D)          (None, 224, 224, 180) 5040
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 180) 0
-----
conv2d_1 (Conv2D)         (None, 112, 112, 128) 207488
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 128) 0
-----
conv2d_2 (Conv2D)         (None, 56, 56, 180) 207540
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 180) 0
-----
conv2d_3 (Conv2D)         (None, 28, 28, 128) 207488
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 128) 0
-----
flatten (Flatten)        (None, 25088)        0
-----
dense (Dense)             (None, 528)          13246992
-----
dropout (Dropout)         (None, 528)          0
-----
dense_1 (Dense)           (None, 528)          279312
-----
dense_2 (Dense)           (None, 250)          132250
=====
Total params: 14,286,110
Trainable params: 14,286,110
Non-trainable params: 0

```

```

-----  

Fitting Model 6 ...  

Epoch 1/10  

551/551 [=====] - 177s 319ms/step - loss: 5.5198 - accuracy: 0.0067 -  

val_loss: 5.5310 - val_accuracy: 0.0040  

Epoch 2/10  

551/551 [=====] - 178s 324ms/step - loss: 5.5085 - accuracy: 0.0092 -  

val_loss: 5.5372 - val_accuracy: 0.0040  

Evaluating:  

40/40 [=====] - 2s 54ms/step - loss: 5.5372 - accuracy: 0.0040  

2 Generation

Model: "cnn"  

-----  

Layer (type)          Output Shape         Param #  

=====  

conv2d (Conv2D)      (None, 224, 224, 41)    1148  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 41)    0  

-----  

conv2d_1 (Conv2D)     (None, 112, 112, 256)   94720  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)   0  

-----  

conv2d_2 (Conv2D)     (None, 56, 56, 180)    414900  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 180)   0  

-----  

conv2d_3 (Conv2D)     (None, 28, 28, 93)     150753  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 93)   0  

-----  

flatten (Flatten)     (None, 18228)        0  

-----  

dense (Dense)         (None, 528)        9624912  

-----  

dropout (Dropout)     (None, 528)        0  

-----  

dense_1 (Dense)       (None, 528)        279312  

-----  

dense_2 (Dense)       (None, 250)        132250  

=====  

Total params: 10,697,995  

Trainable params: 10,697,995  

Non-trainable params: 0  

-----  

Fitting Model 4 ...  

Epoch 1/10  

551/551 [=====] - 180s 323ms/step - loss: 5.4369 - accuracy: 0.0101 -  

val_loss: 4.5853 - val_accuracy: 0.0648  

Epoch 2/10  

551/551 [=====] - 177s 322ms/step - loss: 4.5456 - accuracy: 0.0819 -  

val_loss: 3.6952 - val_accuracy: 0.1840  

Epoch 3/10

```

```

551/551 [=====] - 178s 323ms/step - loss: 3.8351 - accuracy: 0.1801 -
val_loss: 3.2397 - val_accuracy: 0.2544
Epoch 4/10
551/551 [=====] - 177s 322ms/step - loss: 3.3956 - accuracy: 0.2544 -
val_loss: 2.9083 - val_accuracy: 0.3112
Epoch 5/10
551/551 [=====] - 177s 322ms/step - loss: 2.9298 - accuracy: 0.3350 -
val_loss: 2.5219 - val_accuracy: 0.4016
Epoch 6/10
551/551 [=====] - 177s 321ms/step - loss: 2.5772 - accuracy: 0.4023 -
val_loss: 2.2570 - val_accuracy: 0.4368
Epoch 7/10
551/551 [=====] - 177s 322ms/step - loss: 2.2718 - accuracy: 0.4632 -
val_loss: 2.0798 - val_accuracy: 0.4872
Epoch 8/10
551/551 [=====] - 177s 321ms/step - loss: 2.0314 - accuracy: 0.5128 -
val_loss: 1.9965 - val_accuracy: 0.5072
Epoch 9/10
551/551 [=====] - 177s 321ms/step - loss: 1.8160 - accuracy: 0.5637 -
val_loss: 1.9218 - val_accuracy: 0.5184
Epoch 10/10
551/551 [=====] - 177s 321ms/step - loss: 1.6431 - accuracy: 0.5900 -
val_loss: 1.7812 - val_accuracy: 0.5640
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 1.7759 - accuracy: 0.5632
Model: "cnn"

```

Layer ( <code>type</code> )	Output Shape	Param #
conv2d (Conv2D)	( <code>None</code> , 224, 224, 64)	1792
max_pooling2d (MaxPooling2D)	( <code>None</code> , 112, 112, 64)	0
conv2d_1 (Conv2D)	( <code>None</code> , 112, 112, 20)	11540
max_pooling2d_1 (MaxPooling2D)	( <code>None</code> , 56, 56, 20)	0
conv2d_2 (Conv2D)	( <code>None</code> , 56, 56, 200)	36200
max_pooling2d_2 (MaxPooling2D)	( <code>None</code> , 28, 28, 200)	0
conv2d_3 (Conv2D)	( <code>None</code> , 28, 28, 119)	214319
max_pooling2d_3 (MaxPooling2D)	( <code>None</code> , 14, 14, 119)	0
flatten (Flatten)	( <code>None</code> , 23324)	0
dense (Dense)	( <code>None</code> , 528)	12315600
dropout (Dropout)	( <code>None</code> , 528)	0
dense_1 (Dense)	( <code>None</code> , 528)	279312
dense_2 (Dense)	( <code>None</code> , 250)	132250

```

=====
Total params: 12,991,013
Trainable params: 12,991,013
Non-trainable params: 0

-----
Fitting Model 5 ...
Epoch 1/10
551/551 [=====] - 176s 319ms/step - loss: 5.4233 - accuracy: 0.0103 -
val_loss: 4.9168 - val_accuracy: 0.0304
Epoch 2/10
551/551 [=====] - 176s 319ms/step - loss: 4.7715 - accuracy: 0.0526 -
val_loss: 4.1667 - val_accuracy: 0.1360
Epoch 3/10
551/551 [=====] - 175s 318ms/step - loss: 4.1160 - accuracy: 0.1291 -
val_loss: 3.5146 - val_accuracy: 0.2368
Epoch 4/10
551/551 [=====] - 175s 318ms/step - loss: 3.5842 - accuracy: 0.2175 -
val_loss: 3.0911 - val_accuracy: 0.2776
Epoch 5/10
551/551 [=====] - 175s 318ms/step - loss: 3.2091 - accuracy: 0.2835 -
val_loss: 2.8092 - val_accuracy: 0.3416
Epoch 6/10
551/551 [=====] - 175s 318ms/step - loss: 2.8609 - accuracy: 0.3517 -
val_loss: 2.6570 - val_accuracy: 0.3632
Epoch 7/10
551/551 [=====] - 176s 319ms/step - loss: 2.5817 - accuracy: 0.4045 -
val_loss: 2.5177 - val_accuracy: 0.4048
Epoch 8/10
551/551 [=====] - 176s 319ms/step - loss: 2.3541 - accuracy: 0.4444 -
val_loss: 2.3302 - val_accuracy: 0.4360
Epoch 9/10
551/551 [=====] - 175s 318ms/step - loss: 2.1491 - accuracy: 0.4919 -
val_loss: 2.2494 - val_accuracy: 0.4560
Epoch 10/10
551/551 [=====] - 174s 315ms/step - loss: 1.9771 - accuracy: 0.5222 -
val_loss: 2.1461 - val_accuracy: 0.4696
Evaluating:
40/40 [=====] - 2s 51ms/step - loss: 1.9938 - accuracy: 0.5040
Model: "cnn"

-----
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)      (None, 224, 224, 32)   896
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)   0
-----
conv2d_1 (Conv2D)     (None, 112, 112, 49)    14161
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 49)   0
-----
conv2d_2 (Conv2D)     (None, 56, 56, 307)   135694
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 307)   0
-----
```

```

conv2d_3 (Conv2D)           (None, 28, 28, 200)      552800
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)    0
-----
flatten (Flatten)          (None, 39200)            0
-----
dense (Dense)              (None, 528)              20698128
-----
dropout (Dropout)          (None, 528)              0
-----
dense_1 (Dense)            (None, 528)              279312
-----
dense_2 (Dense)            (None, 250)              132250
-----
Total params: 21,813,241
Trainable params: 21,813,241
Non-trainable params: 0
-----
Fitting Model 6 ...
Epoch 1/10
551/551 [=====] - 174s 314ms/step - loss: 5.5203 - accuracy: 0.0063 -
val_loss: 5.5306 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 173s 314ms/step - loss: 5.5087 - accuracy: 0.0097 -
val_loss: 5.5343 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5343 - accuracy: 0.0040
3 Generation

Model: "cnn"
-----
Layer (type)             Output Shape        Param #
=====
conv2d (Conv2D)          (None, 224, 224, 16)   448
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 16)   0
-----
conv2d_1 (Conv2D)         (None, 112, 112, 256)   37120
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)   0
-----
conv2d_2 (Conv2D)         (None, 56, 56, 212)    488660
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 212)   0
-----
conv2d_3 (Conv2D)         (None, 28, 28, 64)     122176
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 64)   0
-----
flatten (Flatten)         (None, 12544)          0
-----
dense (Dense)             (None, 528)            6623760
-----
dropout (Dropout)         (None, 528)            0

```

```

-----  

dense_1 (Dense)           (None, 528)          279312  

-----  

dense_2 (Dense)           (None, 250)          132250  

=====  

Total params: 7,683,726  

Trainable params: 7,683,726  

Non-trainable params: 0  

-----  

Fitting Model 4 ...  

Epoch 1/10  

551/551 [=====] - 175s 316ms/step - loss: 5.5215 - accuracy: 0.0064 -  

val_loss: 5.5304 - val_accuracy: 0.0040  

Epoch 2/10  

551/551 [=====] - 174s 316ms/step - loss: 5.5084 - accuracy: 0.0085 -  

val_loss: 5.5358 - val_accuracy: 0.0040  

Evaluating:  

40/40 [=====] - 2s 51ms/step - loss: 5.5358 - accuracy: 0.0040  

Model: "cnn"  

-----  

Layer (type)          Output Shape        Param #  

=====  

conv2d (Conv2D)         (None, 224, 224, 93)    2604  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 93)    0  

-----  

conv2d_1 (Conv2D)       (None, 112, 112, 256)    214528  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)    0  

-----  

conv2d_2 (Conv2D)       (None, 56, 56, 200)      461000  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 200)    0  

-----  

conv2d_3 (Conv2D)       (None, 28, 28, 83)      149483  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 83)    0  

-----  

flatten (Flatten)       (None, 16268)          0  

-----  

dense (Dense)           (None, 528)          8590032  

-----  

dropout (Dropout)       (None, 528)          0  

-----  

dense_1 (Dense)         (None, 528)          279312  

-----  

dense_2 (Dense)         (None, 250)          132250  

=====  

Total params: 9,829,209  

Trainable params: 9,829,209  

Non-trainable params: 0  

-----  

Fitting Model 5 ...  

Epoch 1/10

```

```

551/551 [=====] - 177s 319ms/step - loss: 5.5207 - accuracy: 0.0053 -
val_loss: 5.5310 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 175s 317ms/step - loss: 5.5069 - accuracy: 0.0074 -
val_loss: 5.5364 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 51ms/step - loss: 5.5364 - accuracy: 0.0040
Model: "cnn"
-----  

Layer (type)          Output Shape         Param #
-----  

conv2d (Conv2D)      (None, 224, 224, 41)    1148  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 41)    0  

-----  

conv2d_1 (Conv2D)     (None, 112, 112, 32)    11840  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 32)    0  

-----  

conv2d_2 (Conv2D)     (None, 56, 56, 228)   65892  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 228)   0  

-----  

conv2d_3 (Conv2D)     (None, 28, 28, 200)   410600  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)   0  

-----  

flatten (Flatten)    (None, 39200)        0  

-----  

dense (Dense)         (None, 528)        20698128  

-----  

dropout (Dropout)    (None, 528)        0  

-----  

dense_1 (Dense)       (None, 528)        279312  

-----  

dense_2 (Dense)       (None, 250)        132250  

-----  

Total params: 21,599,170
Trainable params: 21,599,170
Non-trainable params: 0
-----  

Fitting Model 6 ...
Epoch 1/10
551/551 [=====] - 173s 312ms/step - loss: 5.4638 - accuracy: 0.0109 -
val_loss: 5.0010 - val_accuracy: 0.0320
Epoch 2/10
551/551 [=====] - 172s 313ms/step - loss: 4.8752 - accuracy: 0.0515 -
val_loss: 4.3288 - val_accuracy: 0.1104
Epoch 3/10
551/551 [=====] - 172s 313ms/step - loss: 4.2646 - accuracy: 0.1213 -
val_loss: 3.7418 - val_accuracy: 0.1944
Epoch 4/10
551/551 [=====] - 172s 313ms/step - loss: 3.6774 - accuracy: 0.2036 -
val_loss: 3.0832 - val_accuracy: 0.2944

```

```

Epoch 5/10
551/551 [=====] - 172s 313ms/step - loss: 3.1121 - accuracy: 0.2979 -
val_loss: 2.6061 - val_accuracy: 0.3864
Epoch 6/10
551/551 [=====] - 172s 313ms/step - loss: 2.7114 - accuracy: 0.3752 -
val_loss: 2.3466 - val_accuracy: 0.4536
Epoch 7/10
551/551 [=====] - 172s 313ms/step - loss: 2.4128 - accuracy: 0.4322 -
val_loss: 2.2173 - val_accuracy: 0.4632
Epoch 8/10
551/551 [=====] - 172s 313ms/step - loss: 2.1591 - accuracy: 0.4780 -
val_loss: 2.0380 - val_accuracy: 0.5040
Epoch 9/10
551/551 [=====] - 172s 313ms/step - loss: 1.9425 - accuracy: 0.5326 -
val_loss: 1.8825 - val_accuracy: 0.5432
Epoch 10/10
551/551 [=====] - 172s 312ms/step - loss: 1.7569 - accuracy: 0.5719 -
val_loss: 1.8059 - val_accuracy: 0.5672
Evaluating:
40/40 [=====] - 2s 51ms/step - loss: 1.7910 - accuracy: 0.5568
4 Generation

```

Model: "cnn"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 387)	111843
max_pooling2d_1 (MaxPooling2	(None, 56, 56, 387)	0
conv2d_2 (Conv2D)	(None, 56, 56, 180)	627120
max_pooling2d_2 (MaxPooling2	(None, 28, 28, 180)	0
conv2d_3 (Conv2D)	(None, 28, 28, 62)	100502
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 62)	0
flatten (Flatten)	(None, 12152)	0
dense (Dense)	(None, 528)	6416784
dropout (Dropout)	(None, 528)	0
dense_1 (Dense)	(None, 528)	279312
dense_2 (Dense)	(None, 250)	132250
<hr/>		
Total params: 7,668,707		
Trainable params: 7,668,707		

```

Non-trainable params: 0
-----
Fitting Model 4 ...
Epoch 1/10
551/551 [=====] - 175s 315ms/step - loss: 5.5185 - accuracy: 0.0064 -
val_loss: 5.5299 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 175s 317ms/step - loss: 5.5059 - accuracy: 0.0084 -
val_loss: 5.5324 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5324 - accuracy: 0.0040
Model: "cnn"
-----
Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)      (None, 224, 224, 94)    2632
max_pooling2d (MaxPooling2D) (None, 112, 112, 94)    0
conv2d_1 (Conv2D)      (None, 112, 112, 256)   216832
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)   0
conv2d_2 (Conv2D)      (None, 56, 56, 200)    461000
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 200)   0
conv2d_3 (Conv2D)      (None, 28, 28, 93)     167493
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 93)   0
flatten (Flatten)      (None, 18228)        0
dense (Dense)          (None, 528)        9624912
dropout (Dropout)      (None, 528)        0
dense_1 (Dense)          (None, 528)        279312
dense_2 (Dense)          (None, 250)        132250
=====
Total params: 10,884,431
Trainable params: 10,884,431
Non-trainable params: 0
-----
Fitting Model 5 ...
Epoch 1/10
551/551 [=====] - 176s 317ms/step - loss: 5.5189 - accuracy: 0.0056 -
val_loss: 5.5355 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 174s 316ms/step - loss: 5.5092 - accuracy: 0.0096 -
val_loss: 5.5368 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5368 - accuracy: 0.0040

```

```

Model: "cnn"
-----
Layer (type)          Output Shape       Param #
=====
conv2d (Conv2D)      (None, 224, 224, 41)   1148
max_pooling2d (MaxPooling2D) (None, 112, 112, 41)   0
conv2d_1 (Conv2D)      (None, 112, 112, 28)    10360
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 28)    0
conv2d_2 (Conv2D)      (None, 56, 56, 180)   45540
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 180)   0
conv2d_3 (Conv2D)      (None, 28, 28, 148)   239908
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 148)   0
flatten (Flatten)     (None, 29008)        0
dense (Dense)         (None, 528)        15316752
dropout (Dropout)     (None, 528)        0
dense_1 (Dense)       (None, 528)        279312
dense_2 (Dense)       (None, 250)        132250
=====
Total params: 16,025,270
Trainable params: 16,025,270
Non-trainable params: 0
-----
Fitting Model 6 ...
Epoch 1/10
551/551 [=====] - 173s 312ms/step - loss: 5.5196 - accuracy: 0.0069 -
val_loss: 5.5297 - val_accuracy: 0.0040
Epoch 2/10
551/551 [=====] - 172s 313ms/step - loss: 5.5089 - accuracy: 0.0078 -
val_loss: 5.5333 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5333 - accuracy: 0.0040
5⑨ Geration

```

```

Model: "cnn"
-----
Layer (type)          Output Shape       Param #
=====
conv2d (Conv2D)      (None, 224, 224, 32)   896
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)   0
conv2d_1 (Conv2D)      (None, 112, 112, 307)  88723

```

```

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 307) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 226) 624664  

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 226) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 64) 130240  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64) 0  

-----  

flatten (Flatten) (None, 12544) 0  

-----  

dense (Dense) (None, 528) 6623760  

-----  

dropout (Dropout) (None, 528) 0  

-----  

dense_1 (Dense) (None, 528) 279312  

-----  

dense_2 (Dense) (None, 250) 132250  

=====  

Total params: 7,879,845  

Trainable params: 7,879,845  

Non-trainable params: 0  

-----  

Fitting Model 4 ...  

Epoch 1/10  

551/551 [=====] - 176s 315ms/step - loss: 5.5210 - accuracy: 0.0068 -  

val_loss: 5.5311 - val_accuracy: 0.0040  

Epoch 2/10  

551/551 [=====] - 174s 316ms/step - loss: 5.5060 - accuracy: 0.0081 -  

val_loss: 5.5354 - val_accuracy: 0.0040  

Evaluating:  

40/40 [=====] - 2s 52ms/step - loss: 5.5354 - accuracy: 0.0040  

Model: "cnn"  

-----  

Layer (type) Output Shape Param #  

=====  

conv2d (Conv2D) (None, 224, 224, 50) 1400  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 50) 0  

-----  

conv2d_1 (Conv2D) (None, 112, 112, 256) 115456  

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 256) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 162) 373410  

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 162) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 93) 135687  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 93) 0  

=====
```

flatten (Flatten)	(None, 18228)	0
dense (Dense)	(None, 528)	9624912
dropout (Dropout)	(None, 528)	0
dense_1 (Dense)	(None, 528)	279312
dense_2 (Dense)	(None, 250)	132250
=====		
Total params: 10,662,427		
Trainable params: 10,662,427		
Non-trainable params: 0		
=====		
Fitting Model 5 ...		
Epoch 1/10		
551/551 [=====]	- 177s 319ms/step - loss: 5.4297 - accuracy: 0.0118 -	
val_loss: 4.5592 - val_accuracy: 0.0592		
Epoch 2/10		
551/551 [=====]	- 175s 318ms/step - loss: 4.5245 - accuracy: 0.0834 -	
val_loss: 3.8859 - val_accuracy: 0.1688		
Epoch 3/10		
551/551 [=====]	- 175s 317ms/step - loss: 3.9555 - accuracy: 0.1654 -	
val_loss: 3.4142 - val_accuracy: 0.2272		
Epoch 4/10		
551/551 [=====]	- 175s 318ms/step - loss: 3.4981 - accuracy: 0.2434 -	
val_loss: 3.0268 - val_accuracy: 0.2992		
Epoch 5/10		
551/551 [=====]	- 175s 318ms/step - loss: 3.0417 - accuracy: 0.3134 -	
val_loss: 2.6313 - val_accuracy: 0.3600		
Epoch 6/10		
551/551 [=====]	- 174s 316ms/step - loss: 2.6224 - accuracy: 0.3997 -	
val_loss: 2.2283 - val_accuracy: 0.4368		
Epoch 7/10		
551/551 [=====]	- 175s 317ms/step - loss: 2.2476 - accuracy: 0.4707 -	
val_loss: 2.0325 - val_accuracy: 0.5144		
Epoch 8/10		
551/551 [=====]	- 174s 316ms/step - loss: 1.9440 - accuracy: 0.5348 -	
val_loss: 1.8767 - val_accuracy: 0.5384		
Epoch 9/10		
551/551 [=====]	- 175s 317ms/step - loss: 1.7147 - accuracy: 0.5782 -	
val_loss: 1.7870 - val_accuracy: 0.5680		
Epoch 10/10		
551/551 [=====]	- 175s 317ms/step - loss: 1.5042 - accuracy: 0.6321 -	
val_loss: 1.7390 - val_accuracy: 0.5816		
Evaluating:		
40/40 [=====]	- 2s 52ms/step - loss: 1.7585 - accuracy: 0.5664	
Model: "cnn"	=====	
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 41)	1148
max_pooling2d (MaxPooling2D)	(None, 112, 112, 41)	0

```

-----  

conv2d_1 (Conv2D)           (None, 112, 112, 32)    11840  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 32)    0  

-----  

conv2d_2 (Conv2D)           (None, 56, 56, 172)    49708  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 172)    0  

-----  

conv2d_3 (Conv2D)           (None, 28, 28, 266)    412034  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 266)    0  

-----  

flatten (Flatten)          (None, 52136)        0  

-----  

dense (Dense)              (None, 528)         27528336  

-----  

dropout (Dropout)          (None, 528)         0  

-----  

dense_1 (Dense)            (None, 528)         279312  

-----  

dense_2 (Dense)            (None, 250)         132250  

=====  

Total params: 28,414,628  

Trainable params: 28,414,628  

Non-trainable params: 0  

-----  

Fitting Model 6 ...  

Epoch 1/10  

551/551 [=====] - 174s 315ms/step - loss: 5.3800 - accuracy: 0.0173 -  

val_loss: 4.5299 - val_accuracy: 0.0752  

Epoch 2/10  

551/551 [=====] - 173s 315ms/step - loss: 4.4272 - accuracy: 0.0949 -  

val_loss: 3.9046 - val_accuracy: 0.1600  

Epoch 3/10  

551/551 [=====] - 175s 317ms/step - loss: 3.8455 - accuracy: 0.1796 -  

val_loss: 3.3812 - val_accuracy: 0.2224  

Epoch 4/10  

551/551 [=====] - 175s 318ms/step - loss: 3.5253 - accuracy: 0.2326 -  

val_loss: 3.0517 - val_accuracy: 0.2904  

Epoch 5/10  

551/551 [=====] - 175s 317ms/step - loss: 3.2044 - accuracy: 0.2886 -  

val_loss: 2.8081 - val_accuracy: 0.3504  

Epoch 6/10  

551/551 [=====] - 175s 317ms/step - loss: 2.8866 - accuracy: 0.3464 -  

val_loss: 2.5528 - val_accuracy: 0.3944  

Epoch 7/10  

551/551 [=====] - 175s 318ms/step - loss: 2.6087 - accuracy: 0.3971 -  

val_loss: 2.4190 - val_accuracy: 0.4216  

Epoch 8/10  

551/551 [=====] - 174s 316ms/step - loss: 2.3323 - accuracy: 0.4489 -  

val_loss: 2.3264 - val_accuracy: 0.4496  

Epoch 9/10

```

```

551/551 [=====] - 174s 316ms/step - loss: 2.1520 - accuracy: 0.4838 -
val_loss: 2.2375 - val_accuracy: 0.4832
Epoch 10/10
551/551 [=====] - 174s 316ms/step - loss: 1.9690 - accuracy: 0.5285 -
val_loss: 2.1156 - val_accuracy: 0.5040
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 2.0878 - accuracy: 0.5000
Model: "cnn"
-----  

Layer (type)          Output Shape         Param #
-----  

conv2d (Conv2D)      (None, 224, 224, 50)    1400  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 50)    0  

-----  

conv2d_1 (Conv2D)     (None, 112, 112, 256)   115456  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 256)   0  

-----  

conv2d_2 (Conv2D)     (None, 56, 56, 162)    373410  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 162)   0  

-----  

conv2d_3 (Conv2D)     (None, 28, 28, 93)     135687  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 93)   0  

-----  

flatten (Flatten)    (None, 18228)        0  

-----  

dense (Dense)         (None, 528)        9624912  

-----  

dropout (Dropout)    (None, 528)        0  

-----  

dense_1 (Dense)       (None, 528)        279312  

-----  

dense_2 (Dense)       (None, 250)        132250  

-----  

Total params: 10,662,427
Trainable params: 10,662,427
Non-trainable params: 0
-----  

None

```

## Apêndice B

# Output completo do 2º algoritmo genético

```
(array([[[1, 4, 5, 4],  
       [1, 3, 0, 1],  
       [5, 1, 3, 4],  
       [2, 3, 4, 4],  
       [3, 3, 2, 3],  
       [0, 2, 2, 4]],  
  
      [[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]],  
  
      [[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]],  
  
      [[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]],  
  
      [[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]],  
  
      [[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]]],  
      dtype=uint8), array([4, 4, 4, 4, 4, 4]), array([[[0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0],  
       [0, 0, 0, 0]]], dtype=uint8))  
1º Geration
```

Model: "cnn"

---

Layer (type)	Output Shape	Param #
--------------	--------------	---------

---

```

=====
conv2d (Conv2D)           (None, 224, 224, 64)      1792
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
-----
conv2d_1 (Conv2D)          (None, 112, 112, 200)     115400
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 200)      0
-----
conv2d_2 (Conv2D)          (None, 56, 56, 256)       461056
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 256)      0
-----
conv2d_3 (Conv2D)          (None, 28, 28, 200)       461000
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)      0
-----
flatten (Flatten)          (None, 39200)               0
-----
dense (Dense)              (None, 264)                10349064
-----
dense_1 (Dense)             (None, 264)                69960
-----
dense_2 (Dense)             (None, 250)                66250
=====

Total params: 11,524,522
Trainable params: 11,524,522
Non-trainable params: 0

Fitting Model 1 ...
Epoch 1/5
551/551 [=====] - 177s 319ms/step - loss: 5.2507 - accuracy: 0.0262 -
val_loss: 4.3326 - val_accuracy: 0.1008
Epoch 2/5
551/551 [=====] - 174s 316ms/step - loss: 4.1954 - accuracy: 0.1304 -
val_loss: 3.5758 - val_accuracy: 0.2152
Epoch 3/5
551/551 [=====] - 174s 316ms/step - loss: 3.5630 - accuracy: 0.2273 -
val_loss: 3.1505 - val_accuracy: 0.2856
Epoch 4/5
551/551 [=====] - 174s 316ms/step - loss: 3.1317 - accuracy: 0.3068 -
val_loss: 2.8869 - val_accuracy: 0.3240
Epoch 5/5
551/551 [=====] - 174s 316ms/step - loss: 2.7248 - accuracy: 0.3793 -
val_loss: 2.3950 - val_accuracy: 0.4152
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 2.3443 - accuracy: 0.4320
Model: "cnn"

Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)        (None, 224, 224, 64)      1792
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
=====
```

```

-----  

conv2d_1 (Conv2D)           (None, 112, 112, 180)    103860  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 180)    0  

-----  

conv2d_2 (Conv2D)           (None, 56, 56, 32)      51872  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 32)    0  

-----  

conv2d_3 (Conv2D)           (None, 28, 28, 64)      18496  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 64)    0  

-----  

flatten (Flatten)          (None, 12544)            0  

-----  

dense (Dense)              (None, 264)              3311880  

-----  

dense_1 (Dense)             (None, 264)              69960  

-----  

dense_2 (Dense)             (None, 250)              66250  

=====  

Total params: 3,624,110  

Trainable params: 3,624,110  

Non-trainable params: 0  

-----  

Fitting Model 2 ...  

Epoch 1/5  

551/551 [=====] - 174s 315ms/step - loss: 5.4888 - accuracy: 0.0082 -  

val_loss: 4.6726 - val_accuracy: 0.0664  

Epoch 2/5  

551/551 [=====] - 174s 315ms/step - loss: 4.5879 - accuracy: 0.0837 -  

val_loss: 3.9029 - val_accuracy: 0.1672  

Epoch 3/5  

551/551 [=====] - 174s 315ms/step - loss: 3.8923 - accuracy: 0.1772 -  

val_loss: 3.3773 - val_accuracy: 0.2376  

Epoch 4/5  

551/551 [=====] - 174s 315ms/step - loss: 3.3907 - accuracy: 0.2487 -  

val_loss: 3.0407 - val_accuracy: 0.3152  

Epoch 5/5  

551/551 [=====] - 173s 314ms/step - loss: 2.9268 - accuracy: 0.3347 -  

val_loss: 2.5523 - val_accuracy: 0.3912  

Evaluating:  

40/40 [=====] - 2s 51ms/step - loss: 2.5007 - accuracy: 0.3896  

Model: "cnn"  

-----  

Layer (type)          Output Shape       Param #  

=====  

conv2d (Conv2D)        (None, 224, 224, 256)   7168  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 256)   0  

-----  

conv2d_1 (Conv2D)      (None, 112, 112, 64)     147520  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 64)   0

```

```

-----  

conv2d_2 (Conv2D)           (None, 56, 56, 180)    103860  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 180)    0  

-----  

conv2d_3 (Conv2D)           (None, 28, 28, 200)    324200  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)    0  

-----  

flatten (Flatten)          (None, 39200)        0  

-----  

dense (Dense)              (None, 264)         10349064  

-----  

dense_1 (Dense)            (None, 264)         69960  

-----  

dense_2 (Dense)            (None, 250)         66250  

=====  

Total params: 11,068,022  

Trainable params: 11,068,022  

Non-trainable params: 0  

-----  

Fitting Model 3 ...  

Epoch 1/5  

551/551 [=====] - 176s 318ms/step - loss: 5.5249 - accuracy: 0.0081 -  

val_loss: 5.5274 - val_accuracy: 0.0040  

Epoch 2/5  

551/551 [=====] - 177s 320ms/step - loss: 5.5091 - accuracy: 0.0091 -  

val_loss: 5.5343 - val_accuracy: 0.0040  

Evaluating:  

40/40 [=====] - 2s 53ms/step - loss: 5.5343 - accuracy: 0.0040  

Model: "cnn"  

-----  

Layer (type)             Output Shape        Param #  

=====  

conv2d (Conv2D)          (None, 224, 224, 128)   3584  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 128)   0  

-----  

conv2d_1 (Conv2D)         (None, 112, 112, 180)   207540  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 180)   0  

-----  

conv2d_2 (Conv2D)         (None, 56, 56, 200)    324200  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 200)   0  

-----  

conv2d_3 (Conv2D)         (None, 28, 28, 200)    360200  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)   0  

-----  

flatten (Flatten)         (None, 39200)        0  

-----  

dense (Dense)             (None, 264)         10349064  

-----
```

```

dense_1 (Dense)           (None, 264)          69960
-----
dense_2 (Dense)           (None, 250)          66250
=====
Total params: 11,380,798
Trainable params: 11,380,798
Non-trainable params: 0
-----
Fitting Model 4 ...
Epoch 1/5
551/551 [=====] - 178s 321ms/step - loss: 5.5223 - accuracy: 0.0088 -
val_loss: 5.5274 - val_accuracy: 0.0040
Epoch 2/5
551/551 [=====] - 176s 319ms/step - loss: 5.5058 - accuracy: 0.0091 -
val_loss: 5.5344 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 5.5344 - accuracy: 0.0040
Model: "cnn"
-----
Layer (type)             Output Shape        Param #
=====
conv2d (Conv2D)          (None, 224, 224, 180) 5040
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 180) 0
-----
conv2d_1 (Conv2D)         (None, 112, 112, 180) 291780
-----
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 180) 0
-----
conv2d_2 (Conv2D)         (None, 56, 56, 128) 207488
-----
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 128) 0
-----
conv2d_3 (Conv2D)         (None, 28, 28, 180) 207540
-----
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 180) 0
-----
flatten (Flatten)         (None, 35280)        0
-----
dense (Dense)             (None, 264)          9314184
-----
dense_1 (Dense)           (None, 264)          69960
-----
dense_2 (Dense)           (None, 250)          66250
=====
Total params: 10,162,242
Trainable params: 10,162,242
Non-trainable params: 0
-----
Fitting Model 5 ...
Epoch 1/5
551/551 [=====] - 178s 320ms/step - loss: 5.5202 - accuracy: 0.0072 -
val_loss: 5.5261 - val_accuracy: 0.0040
Epoch 2/5

```

```

551/551 [=====] - 175s 318ms/step - loss: 5.5076 - accuracy: 0.0080 -
val_loss: 5.5326 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 5.5326 - accuracy: 0.0040
Model: "cnn"
-----  

Layer (type)          Output Shape         Param #
-----  

conv2d (Conv2D)       (None, 224, 224, 32)    896  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 32)    0  

-----  

conv2d_1 (Conv2D)      (None, 112, 112, 128)   36992  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 128)    0  

-----  

conv2d_2 (Conv2D)      (None, 56, 56, 128)    147584  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 128)    0  

-----  

conv2d_3 (Conv2D)      (None, 28, 28, 200)   230600  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)    0  

-----  

flatten (Flatten)      (None, 39200)        0  

-----  

dense (Dense)          (None, 264)        10349064  

-----  

dense_1 (Dense)        (None, 264)        69960  

-----  

dense_2 (Dense)        (None, 250)        66250  

-----  

Total params: 10,901,346
Trainable params: 10,901,346
Non-trainable params: 0
-----  

Fitting Model 6 ...
Epoch 1/5
551/551 [=====] - 174s 314ms/step - loss: 5.4220 - accuracy: 0.0133 -
val_loss: 4.8891 - val_accuracy: 0.0440
Epoch 2/5
551/551 [=====] - 174s 316ms/step - loss: 4.7586 - accuracy: 0.0625 -
val_loss: 4.0605 - val_accuracy: 0.1432
Epoch 3/5
551/551 [=====] - 174s 316ms/step - loss: 3.9540 - accuracy: 0.1650 -
val_loss: 3.2656 - val_accuracy: 0.2656
Epoch 4/5
551/551 [=====] - 175s 318ms/step - loss: 3.2845 - accuracy: 0.2743 -
val_loss: 2.7668 - val_accuracy: 0.3528
Epoch 5/5
551/551 [=====] - 174s 316ms/step - loss: 2.6833 - accuracy: 0.3800 -
val_loss: 2.2234 - val_accuracy: 0.4528
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 2.2487 - accuracy: 0.4416

```

## 2. Generation

```
Model: "cnn"

Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)       (None, 224, 224, 29)    812
max_pooling2d (MaxPooling2D) (None, 112, 112, 29)    0
conv2d_1 (Conv2D)      (None, 112, 112, 208)   54496
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 208)    0
conv2d_2 (Conv2D)      (None, 56, 56, 128)    239744
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 128)    0
conv2d_3 (Conv2D)      (None, 28, 28, 200)   230600
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 200)    0
flatten (Flatten)     (None, 39200)        0
dense (Dense)          (None, 264)        10349064
dense_1 (Dense)        (None, 264)        69960
dense_2 (Dense)        (None, 250)        66250
=====
Total params: 11,010,926
Trainable params: 11,010,926
Non-trainable params: 0

Fitting Model 4 ...
Epoch 1/5
551/551 [=====] - 177s 319ms/step - loss: 5.3394 - accuracy: 0.0148 -
val_loss: 4.4885 - val_accuracy: 0.0720
Epoch 2/5
551/551 [=====] - 175s 318ms/step - loss: 4.3854 - accuracy: 0.0991 -
val_loss: 3.6309 - val_accuracy: 0.1760
Epoch 3/5
551/551 [=====] - 175s 318ms/step - loss: 3.6267 - accuracy: 0.2084 -
val_loss: 3.0568 - val_accuracy: 0.2744
Epoch 4/5
551/551 [=====] - 176s 319ms/step - loss: 3.0640 - accuracy: 0.3137 -
val_loss: 2.5359 - val_accuracy: 0.4064
Epoch 5/5
551/551 [=====] - 175s 318ms/step - loss: 2.5321 - accuracy: 0.4100 -
val_loss: 2.1543 - val_accuracy: 0.4744
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 2.0385 - accuracy: 0.4880
Model: "cnn"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 89)	2492
max_pooling2d (MaxPooling2D)	(None, 112, 112, 89)	0
conv2d_1 (Conv2D)	(None, 112, 112, 180)	144360
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 180)	0
conv2d_2 (Conv2D)	(None, 56, 56, 256)	414976
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_3 (Conv2D)	(None, 28, 28, 28)	64540
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 28)	0
flatten (Flatten)	(None, 5488)	0
dense (Dense)	(None, 264)	1449096
dense_1 (Dense)	(None, 264)	69960
dense_2 (Dense)	(None, 250)	66250
Total params:	2,211,674	
Trainable params:	2,211,674	
Non-trainable params:	0	
Fitting Model 5 ...		
Epoch 1/5		
551/551 [=====] - 178s 320ms/step - loss: 5.5225 - accuracy: 0.0070 - val_loss: 5.5270 - val_accuracy: 0.0040		
Epoch 2/5		
551/551 [=====] - 176s 320ms/step - loss: 5.5078 - accuracy: 0.0077 - val_loss: 5.5322 - val_accuracy: 0.0040		
Evaluating:		
40/40 [=====] - 2s 52ms/step - loss: 5.5322 - accuracy: 0.0040		
Model: "cnn"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_1 (Conv2D)	(None, 112, 112, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_2 (Conv2D)	(None, 56, 56, 39)	44967
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 39)	0

```

-----  

conv2d_3 (Conv2D)           (None, 28, 28, 282)    99264  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 282)    0  

-----  

flatten (Flatten)           (None, 55272)        0  

-----  

dense (Dense)               (None, 264)          14592072  

-----  

dense_1 (Dense)             (None, 264)          69960  

-----  

dense_2 (Dense)             (None, 250)          66250  

-----  

Total params: 14,948,161  

Trainable params: 14,948,161  

Non-trainable params: 0  

-----  

Fitting Model 6 ...  

Epoch 1/5  

551/551 [=====] - 176s 319ms/step - loss: 5.4601 - accuracy: 0.0099 -  

val_loss: 4.6024 - val_accuracy: 0.0488  

Epoch 2/5  

551/551 [=====] - 176s 319ms/step - loss: 4.5062 - accuracy: 0.0862 -  

val_loss: 3.8681 - val_accuracy: 0.1592  

Epoch 3/5  

551/551 [=====] - 176s 320ms/step - loss: 3.9144 - accuracy: 0.1679 -  

val_loss: 3.4601 - val_accuracy: 0.2344  

Epoch 4/5  

551/551 [=====] - 176s 319ms/step - loss: 3.4894 - accuracy: 0.2396 -  

val_loss: 3.0355 - val_accuracy: 0.3112  

Epoch 5/5  

551/551 [=====] - 176s 319ms/step - loss: 2.9897 - accuracy: 0.3225 -  

val_loss: 2.6456 - val_accuracy: 0.3656  

Evaluating:  

40/40 [=====] - 2s 52ms/step - loss: 2.5652 - accuracy: 0.3808  

39 Generation  

-----  

Model: "cnn"  

-----  

Layer (type)          Output Shape       Param #  

-----  

conv2d (Conv2D)           (None, 224, 224, 29)    812  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 29)    0  

-----  

conv2d_1 (Conv2D)          (None, 112, 112, 128)    33536  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 128)    0  

-----  

conv2d_2 (Conv2D)          (None, 56, 56, 107)      123371  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 107)    0  

-----  

conv2d_3 (Conv2D)          (None, 28, 28, 299)      288236

```

```

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 299) 0
-----
flatten (Flatten) (None, 58604) 0
-----
dense (Dense) (None, 264) 15471720
-----
dense_1 (Dense) (None, 264) 69960
-----
dense_2 (Dense) (None, 250) 66250
=====
Total params: 16,053,885
Trainable params: 16,053,885
Non-trainable params: 0
-----
Fitting Model 4 ...
Epoch 1/5
551/551 [=====] - 176s 318ms/step - loss: 5.2975 - accuracy: 0.0197 -
val_loss: 4.2893 - val_accuracy: 0.1048
Epoch 2/5
551/551 [=====] - 175s 318ms/step - loss: 4.2264 - accuracy: 0.1197 -
val_loss: 3.3886 - val_accuracy: 0.2280
Epoch 3/5
551/551 [=====] - 175s 318ms/step - loss: 3.4432 - accuracy: 0.2392 -
val_loss: 2.8239 - val_accuracy: 0.3272
Epoch 4/5
551/551 [=====] - 175s 318ms/step - loss: 2.8491 - accuracy: 0.3451 -
val_loss: 2.4060 - val_accuracy: 0.4152
Epoch 5/5
551/551 [=====] - 175s 317ms/step - loss: 2.3747 - accuracy: 0.4444 -
val_loss: 2.0527 - val_accuracy: 0.5048
Evaluating:
40/40 [=====] - 2s 51ms/step - loss: 2.0025 - accuracy: 0.4744
Model: "cnn"
-----
Layer (type) Output Shape Param #
=====
conv2d (Conv2D) (None, 224, 224, 26) 728
-----
max_pooling2d (MaxPooling2D) (None, 112, 112, 26) 0
-----
conv2d_1 (Conv2D) (None, 112, 112, 200) 47000
-----
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 200) 0
-----
conv2d_2 (Conv2D) (None, 56, 56, 128) 230528
-----
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128) 0
-----
conv2d_3 (Conv2D) (None, 28, 28, 202) 232906
-----
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 202) 0
-----
flatten (Flatten) (None, 39592) 0

```

```

-----  

dense (Dense)           (None, 264)      10452552  

-----  

dense_1 (Dense)          (None, 264)      69960  

-----  

dense_2 (Dense)          (None, 250)      66250  

=====  

Total params: 11,099,924  

Trainable params: 11,099,924  

Non-trainable params: 0  

-----  

Fitting Model 5 ...  

Epoch 1/5  

551/551 [=====] - 176s 318ms/step - loss: 5.5017 - accuracy: 0.0073 -  

val_loss: 4.9754 - val_accuracy: 0.0376  

Epoch 2/5  

551/551 [=====] - 175s 318ms/step - loss: 4.7108 - accuracy: 0.0649 -  

val_loss: 3.7452 - val_accuracy: 0.1832  

Epoch 3/5  

551/551 [=====] - 176s 319ms/step - loss: 3.7598 - accuracy: 0.1911 -  

val_loss: 3.1501 - val_accuracy: 0.3088  

Epoch 4/5  

551/551 [=====] - 176s 319ms/step - loss: 3.1596 - accuracy: 0.2869 -  

val_loss: 2.5844 - val_accuracy: 0.3776  

Epoch 5/5  

551/551 [=====] - 176s 319ms/step - loss: 2.6637 - accuracy: 0.3849 -  

val_loss: 2.1775 - val_accuracy: 0.4544  

Evaluating:  

40/40 [=====] - 2s 52ms/step - loss: 2.1255 - accuracy: 0.4656  

Model: "cnn"  

-----  

Layer (type)          Output Shape       Param #  

=====  

conv2d (Conv2D)        (None, 224, 224, 46)    1288  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 46)    0  

-----  

conv2d_1 (Conv2D)       (None, 112, 112, 208)    86320  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 208)    0  

-----  

conv2d_2 (Conv2D)       (None, 56, 56, 256)     479488  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 256)    0  

-----  

conv2d_3 (Conv2D)       (None, 28, 28, 105)     242025  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 105)    0  

-----  

flatten (Flatten)       (None, 20580)         0  

-----  

dense (Dense)           (None, 264)      5433384  

-----  

dense_1 (Dense)          (None, 264)      69960

```

```

dense_2 (Dense)           (None, 250)          66250
=====
Total params: 6,378,715
Trainable params: 6,378,715
Non-trainable params: 0

Fitting Model 6 ...
Epoch 1/5
551/551 [=====] - 178s 321ms/step - loss: 5.5198 - accuracy: 0.0046 -
val_loss: 5.5309 - val_accuracy: 0.0040
Epoch 2/5
551/551 [=====] - 177s 320ms/step - loss: 5.5089 - accuracy: 0.0074 -
val_loss: 5.5317 - val_accuracy: 0.0040
Evaluating:
40/40 [=====] - 2s 52ms/step - loss: 5.5317 - accuracy: 0.0040
4 Generation

Model: "cnn"

Layer (type)             Output Shape        Param #
=====
conv2d (Conv2D)          (None, 224, 224, 12)   336
max_pooling2d (MaxPooling2D) (None, 112, 112, 12)   0
conv2d_1 (Conv2D)         (None, 112, 112, 128)  13952
max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 128)  0
conv2d_2 (Conv2D)         (None, 56, 56, 116)   133748
max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 116)  0
conv2d_3 (Conv2D)         (None, 28, 28, 299)  312455
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 299)  0
flatten (Flatten)        (None, 58604)        0
dense (Dense)            (None, 264)          15471720
dense_1 (Dense)          (None, 264)          69960
dense_2 (Dense)          (None, 250)          66250
=====
Total params: 16,068,421
Trainable params: 16,068,421
Non-trainable params: 0

Fitting Model 4 ...
Epoch 1/5
551/551 [=====] - 176s 318ms/step - loss: 5.3576 - accuracy: 0.0171 -
val_loss: 4.4919 - val_accuracy: 0.0680

```

```

Epoch 2/5
551/551 [=====] - 176s 319ms/step - loss: 4.4582 - accuracy: 0.0969 -
val_loss: 3.8383 - val_accuracy: 0.1672
Epoch 3/5
551/551 [=====] - 176s 319ms/step - loss: 3.9056 - accuracy: 0.1759 -
val_loss: 3.4436 - val_accuracy: 0.2360
Epoch 4/5
551/551 [=====] - 176s 319ms/step - loss: 3.4128 - accuracy: 0.2543 -
val_loss: 2.8870 - val_accuracy: 0.3240
Epoch 5/5
551/551 [=====] - 176s 319ms/step - loss: 2.8879 - accuracy: 0.3440 -
val_loss: 2.5254 - val_accuracy: 0.4000
Evaluating:
40/40 [=====] - 2s 53ms/step - loss: 2.4802 - accuracy: 0.4040
Model: "cnn"
-----  

Layer (type)          Output Shape         Param #
-----  

conv2d (Conv2D)      (None, 224, 224, 17)    476  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 17)    0  

-----  

conv2d_1 (Conv2D)     (None, 112, 112, 200)   30800  

-----  

max_pooling2d_1 (MaxPooling2D) (None, 56, 56, 200)   0  

-----  

conv2d_2 (Conv2D)     (None, 56, 56, 107)    192707  

-----  

max_pooling2d_2 (MaxPooling2D) (None, 28, 28, 107)   0  

-----  

conv2d_3 (Conv2D)     (None, 28, 28, 306)    294984  

-----  

max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 306)   0  

-----  

flatten (Flatten)    (None, 59976)        0  

-----  

dense (Dense)        (None, 264)        15833928  

-----  

dense_1 (Dense)      (None, 264)        69960  

-----  

dense_2 (Dense)      (None, 250)        66250  

-----  

Total params: 16,489,105
Trainable params: 16,489,105
Non-trainable params: 0
-----  

Fitting Model 5 ...
Epoch 1/5
551/551 [=====] - 177s 320ms/step - loss: 5.5193 - accuracy: 0.0059 -
val_loss: 5.5294 - val_accuracy: 0.0040
Epoch 2/5
551/551 [=====] - 175s 318ms/step - loss: 5.5088 - accuracy: 0.0083 -
val_loss: 5.5342 - val_accuracy: 0.0040
Evaluating:

```

```
40/40 [=====] - 2s 52ms/step - loss: 5.5342 - accuracy: 0.0040
Model: "cnn"
```

Layer ( <code>type</code> )	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 26)	728
max_pooling2d (MaxPooling2D)	(None, 112, 112, 26)	0
conv2d_1 (Conv2D)	(None, 112, 112, 208)	48880
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 208)	0
conv2d_2 (Conv2D)	(None, 56, 56, 88)	164824
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 88)	0
conv2d_3 (Conv2D)	(None, 28, 28, 179)	141947
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 179)	0
flatten (Flatten)	(None, 35084)	0
dense (Dense)	(None, 264)	9262440
dense_1 (Dense)	(None, 264)	69960
dense_2 (Dense)	(None, 250)	66250

```
Total params: 9,755,029
```

```
Trainable params: 9,755,029
```

```
Non-trainable params: 0
```

---

```
Fitting Model 6 ...
```

```
Epoch 1/5
```

```
551/551 [=====] - 177s 319ms/step - loss: 5.4754 - accuracy: 0.0084 -  
val_loss: 4.6961 - val_accuracy: 0.0664
```

```
Epoch 2/5
```

```
551/551 [=====] - 176s 319ms/step - loss: 4.6044 - accuracy: 0.0759 -  
val_loss: 4.0139 - val_accuracy: 0.1288
```

```
Epoch 3/5
```

```
551/551 [=====] - 175s 318ms/step - loss: 4.0065 - accuracy: 0.1521 -  
val_loss: 3.5199 - val_accuracy: 0.2008
```

```
Epoch 4/5
```

```
551/551 [=====] - 175s 317ms/step - loss: 3.5835 - accuracy: 0.2178 -  
val_loss: 3.1327 - val_accuracy: 0.2800
```

```
Epoch 5/5
```

```
551/551 [=====] - 176s 319ms/step - loss: 3.1410 - accuracy: 0.2996 -  
val_loss: 2.7763 - val_accuracy: 0.3568
```

```
Evaluating:
```

```
40/40 [=====] - 2s 53ms/step - loss: 2.7670 - accuracy: 0.3568
```

```
5 Generation
```

```
Model: "cnn"
```

Layer ( <code>type</code> )	Output Shape	Param #
conv2d (Conv2D)	( <code>None</code> , 224, 224, 27)	756
max_pooling2d (MaxPooling2D)	( <code>None</code> , 112, 112, 27)	0
conv2d_1 (Conv2D)	( <code>None</code> , 112, 112, 128)	31232
max_pooling2d_1 (MaxPooling2D)	( <code>None</code> , 56, 56, 128)	0
conv2d_2 (Conv2D)	( <code>None</code> , 56, 56, 111)	127983
max_pooling2d_2 (MaxPooling2D)	( <code>None</code> , 28, 28, 111)	0
conv2d_3 (Conv2D)	( <code>None</code> , 28, 28, 299)	299000
max_pooling2d_3 (MaxPooling2D)	( <code>None</code> , 14, 14, 299)	0
flatten (Flatten)	( <code>None</code> , 58604)	0
dense (Dense)	( <code>None</code> , 264)	15471720
dense_1 (Dense)	( <code>None</code> , 264)	69960
dense_2 (Dense)	( <code>None</code> , 250)	66250

Total params: 16,066,901  
Trainable params: 16,066,901  
Non-trainable params: 0

---

Fitting Model 4 ...

Epoch 1/5  
551/551 [=====] - 177s 319ms/step - loss: 5.4817 - accuracy: 0.0084 -  
val\_loss: 4.7724 - val\_accuracy: 0.0488

Epoch 2/5  
551/551 [=====] - 175s 318ms/step - loss: 4.6422 - accuracy: 0.0687 -  
val\_loss: 3.8720 - val\_accuracy: 0.1520

Epoch 3/5  
551/551 [=====] - 175s 318ms/step - loss: 3.8276 - accuracy: 0.1773 -  
val\_loss: 3.1990 - val\_accuracy: 0.2640

Epoch 4/5  
551/551 [=====] - 175s 318ms/step - loss: 3.2585 - accuracy: 0.2711 -  
val\_loss: 2.6894 - val\_accuracy: 0.3448

Epoch 5/5  
551/551 [=====] - 175s 317ms/step - loss: 2.7559 - accuracy: 0.3627 -  
val\_loss: 2.3018 - val\_accuracy: 0.4440

Evaluating:

40/40 [=====] - 2s 53ms/step - loss: 2.2416 - accuracy: 0.4616

Model: "cnn"

---

Layer ( <code>type</code> )	Output Shape	Param #
conv2d (Conv2D)	( <code>None</code> , 224, 224, 29)	812

```

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 29) 0  

-----  

conv2d_1 (Conv2D) (None, 112, 112, 187) 48994  

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 187) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 107) 180188  

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 107) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 108) 104112  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 108) 0  

-----  

flatten (Flatten) (None, 21168) 0  

-----  

dense (Dense) (None, 264) 5588616  

-----  

dense_1 (Dense) (None, 264) 69960  

-----  

dense_2 (Dense) (None, 250) 66250  

=====  

Total params: 6,058,932  

Trainable params: 6,058,932  

Non-trainable params: 0  

-----  

Fitting Model 5 ...  

Epoch 1/5  

551/551 [=====] - 176s 317ms/step - loss: 5.5143 - accuracy: 0.0081 -  

val_loss: 5.1012 - val_accuracy: 0.0248  

Epoch 2/5  

551/551 [=====] - 175s 318ms/step - loss: 4.7450 - accuracy: 0.0581 -  

val_loss: 3.7899 - val_accuracy: 0.1560  

Epoch 3/5  

551/551 [=====] - 175s 317ms/step - loss: 3.7837 - accuracy: 0.1815 -  

val_loss: 3.2222 - val_accuracy: 0.2520  

Epoch 4/5  

551/551 [=====] - 175s 317ms/step - loss: 3.2771 - accuracy: 0.2697 -  

val_loss: 2.7097 - val_accuracy: 0.3536  

Epoch 5/5  

551/551 [=====] - 174s 316ms/step - loss: 2.8026 - accuracy: 0.3525 -  

val_loss: 2.3578 - val_accuracy: 0.4240  

Evaluating:  

40/40 [=====] - 2s 53ms/step - loss: 2.2957 - accuracy: 0.4368  

Model: "cnn"  

-----  

Layer (type) Output Shape Param #  

=====  

conv2d (Conv2D) (None, 224, 224, 36) 1008  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 36) 0  

-----  

conv2d_1 (Conv2D) (None, 112, 112, 171) 55575

```

```

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 171) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 128) 197120  

-----  

max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128) 0  

-----  

conv2d_3 (Conv2D) (None, 28, 28, 200) 230600  

-----  

max_pooling2d_3 (MaxPooling2 (None, 14, 14, 200) 0  

-----  

flatten (Flatten) (None, 39200) 0  

-----  

dense (Dense) (None, 264) 10349064  

-----  

dense_1 (Dense) (None, 264) 69960  

-----  

dense_2 (Dense) (None, 250) 66250  

=====  

Total params: 10,969,577  

Trainable params: 10,969,577  

Non-trainable params: 0  

-----  

Fitting Model 6 ...  

Epoch 1/5  

551/551 [=====] - 177s 319ms/step - loss: 5.5209 - accuracy: 0.0064 -  

val_loss: 5.5297 - val_accuracy: 0.0040  

Epoch 2/5  

551/551 [=====] - 176s 320ms/step - loss: 5.4351 - accuracy: 0.0139 -  

val_loss: 4.5442 - val_accuracy: 0.0744  

Epoch 3/5  

551/551 [=====] - 176s 319ms/step - loss: 4.4956 - accuracy: 0.0906 -  

val_loss: 3.8262 - val_accuracy: 0.1680  

Epoch 4/5  

551/551 [=====] - 175s 318ms/step - loss: 3.8902 - accuracy: 0.1698 -  

val_loss: 3.3204 - val_accuracy: 0.2456  

Epoch 5/5  

551/551 [=====] - 175s 317ms/step - loss: 3.4025 - accuracy: 0.2463 -  

val_loss: 2.8332 - val_accuracy: 0.3168  

Evaluating:  

40/40 [=====] - 2s 53ms/step - loss: 2.7950 - accuracy: 0.3360  

Model: "cnn"  

-----  

Layer (type) Output Shape Param #  

=====  

conv2d (Conv2D) (None, 224, 224, 27) 756  

-----  

max_pooling2d (MaxPooling2D) (None, 112, 112, 27) 0  

-----  

conv2d_1 (Conv2D) (None, 112, 112, 128) 31232  

-----  

max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128) 0  

-----  

conv2d_2 (Conv2D) (None, 56, 56, 111) 127983

```

```
-----  
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 111) 0  
-----  
conv2d_3 (Conv2D) (None, 28, 28, 299) 299000  
-----  
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 299) 0  
-----  
flatten (Flatten) (None, 58604) 0  
-----  
dense (Dense) (None, 264) 15471720  
-----  
dense_1 (Dense) (None, 264) 69960  
-----  
dense_2 (Dense) (None, 250) 66250  
=====  
Total params: 16,066,901  
Trainable params: 16,066,901  
Non-trainable params: 0  
-----
```