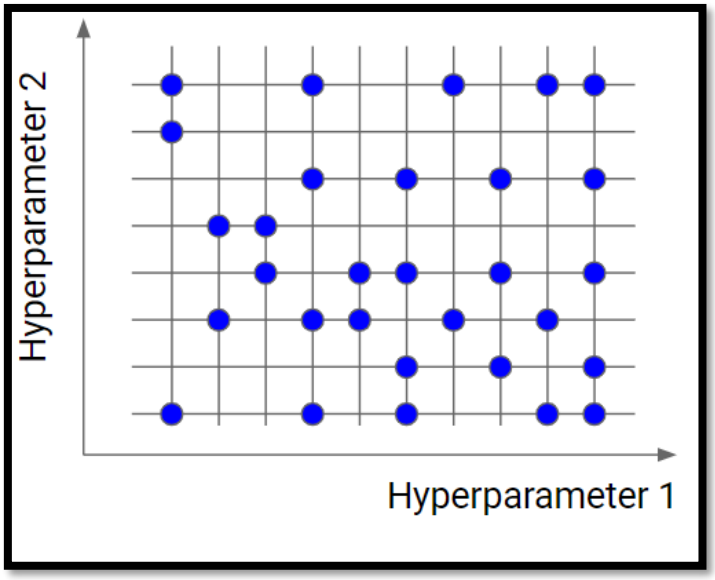
Departamento de Engenharia Informática e de Sistemas

Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Inteligência Computacional 2022/2023

Projeto Fase - 3



Emanuel Saraiva | 2019130219 | a2019130219@isec.pt | P1

Índice

2. Descrição do problema

Como se sabe, o problema desta 3ª fase do trabalho é o mesmo da 1ª fase. Desta forma apresento a mesma descrição usada anteriormente:

Nos dias de hoje uma das principais preocupações a nível mundial são as alterações climáticas (climate change). A principal causa das alterações climáticas é a combustão de combustíveis fósseis como o petróleo, o carvão e o gás natural, que emitem gases com efeito de estufa para a atmosfera, provocando assim a alteração dos níveis do mar, a variação de temperatura e a precipitação inesperada. Cada vez mais se optam por medidas de reciclagem para evitar as lixeiras em céu aberto que emitem gases poluentes e consequentemente causam o aumento da temperatura da terra.

O nosso *dataset* não é balanceado, pois existem 3 tipos de vidro (castanho, transparente e verde) e existem 5 mil exemplos de roupa o que nas restantes classes rondam os mil exemplos.

Este *dataset* é um problema de classificação, com recurso a 15.5 mil imagens distribuídas por 10 classes distintas.

As 10 classes são:

- Papel;
- Vidro;
- Cartão;
- Roupa;
- Sapatos;
- Metal;
- Lixo;
- Biológico;
- Bateria;
- Plástico.

Nota: O *dataset* era constituído por 12 classes, no entanto juntámos os três tipos de vidro de maneira a ter apenas uma classe, acelerando assim o processo de treino.

Link do *dataset*: https://www.kaggle.com/datasets/mostafaabla/garbage-classification.

3. Descrição das metodologias e algoritmos utilizados

Esta fase é dividida em três partes. Na primeira, usei o *python* com recurso às seguintes bibliotecas:

- Keras (é também uma API);
- Scikit-Learn;
- TensorFlow;
- Numpy
- Pandas;
- Matplotlib;
- Re;
- Entre outras.

Para testar diferentes Hiper-parâmetros e Camadas.

Estes foram os testes realizados:

Camadas:

- A "**mobilenetv2**_layer" guarda os dados da camada MobileNetV2. A MobileNetV2 usa a arquitetura do modelo CNN para classificar imagens. A vantagem de a usar é que como já está pré-treinada, já não precisamos de treinar a rede de raiz;
- A função da camada "**GlobalAveragePooling2D**" é aplicar o "pooling" médio (é o cálculo médio para cada feature map – é um mapa onde um certo tipo de feature é encontrado na imagem) nas dimensões espaciais até que cada dimensão espacial esteja ativada;
- A "**Flatten**" transforma um array bidimensional num vetor. Decidimos usá-la, pois estávamos curiosos com a possível diferença entre estas duas;
- A "**Dropout**" é a camada que vai tratar o "overfitting". Esta certifica-se que o modelo consegue usar bem as imagens de treino e de teste. Leva como argumento, um número, o qual corresponde à percentagem de neurónios que vão ser retirados de forma aleatória no treino para controlar o overfitting;

- A camada "**Dense**" é a camada de output (saída) – tem 10 neurónios porque o nosso dataset tem 10 classes (len(categories)).

Funções de Ativação:

Decidimos usar a "**Softmax**" por ser a mais usada neste tipo de situação.

Vimos na internet que a "**Relu**" era conhecida por ser mais rápida, então decidimos experimentar.

Código para construir a rede neuronal:

```
# mobilenetv2 é um modelo (deep learning) do tipo CNN que é pré-treinado, ou seja,
# já foi treinada para classificar imagens, sendo esta capaz de classificar
# 1000 categorias das mesmas
mobilenetv2_layer = mobilenetv2.MobileNetV2(include_top = False, input_shape = (IMAGE_WIDTH, IMAGE_HEIGHT,IMAGE_CHA
                        weights = 'imagenet')

# O modelo mobilenetv2 já está pré-treinado, logo, não queremos que estas camadas que ele tras
# sejam treinadas novamente pelo tensorflow
mobilenetv2_layer.trainable = False

model = Sequential()

model.add(keras.Input(shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))

# Cria uma camada para aplicar o preprocessamento na imagem, para ter as caracteristicas pretendidas
def mobilenetv2_preprocessing(img):
  return mobilenetv2.preprocess_input(img)

model.add(Lambda(mobilenetv2_preprocessing))

#model.add(tf.keras.layers.BatchNormalization()) | Demora mt tempo a treinar com esta camada +/- 15 por epoch

# Camadas pre treinadas
model.add(mobilenetv2_layer)


model.add(tf.keras.layers.GlobalAveragePooling2D())
model.add(Flatten(name="featuresCamadaFlatten"))
model.add(tf.keras.layers.Dropout(0.3))
model.add(Dense(len(categories), activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['categorical_accuracy'])
model.summary()


feature_extractor = keras.Model(
    inputs=model.inputs,
    outputs=model.get_layer(name="featuresCamadaFlatten").output,
)

x = tf.ones((1, 224, 224, 3))
features = feature_extractor(x)
print("Número de Features da Camada featuresCamadaFlatten:")
print(features)
```

Resultado do código:

```
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 featuresCamadaGlobal (Globa  (None, 1280)             0
 lAveragePooling2D)

 dense (Dense)               (None, 10)                12810

=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
_________________________________________________________________
Número de Features da Camada GlobalAveragePooling2D:
tf.Tensor([[0.55155    0.         0.44229084 ... 0.         0.4064743  0.12210276]], shape=(1, 1280), dtype=float3
2)
```

Decidimos dividir o nosso dataset em 80% treino, 10% validação e 10% teste, de modo a ter dados mais fidedignos e não tão tendenciosos.

Ao fazermos esta divisão, estamos a prevenir que o nosso modelo entre em overfitting, para além de que ao ser gerado os dados sobre a validação, conseguimos combater de igual forma o overfitting.

Código da divisão do dataset:

```
# Muda as categorias de numeros para nomes
df["category"] = df["category"].replace(categories)

# Dividimos o nosso dataset em treino, validação e teste, de modo a ter dados mais fidedignos e não
# tão tendenciosos. 80% para treino, 10% para validação e 10% para teste
# Ao dividir em validação permite-nos ver se o modelo está em overfitting, gera-nos outros dados que não
# os de treino
train_df, validate_df = train_test_split(df, test_size=0.2, random_state=42)
validate_df, test_df = train_test_split(validate_df, test_size=0.5, random_state=42)

train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)

total_train = train_df.shape[0]
total_validate = validate_df.shape[0]

print('Num Imagens de Treino = ', total_train , '  Num Imagens de Validação = ', total_validate, '  Num Imagens de
```

Resultado:

```
Num Imagens de Treino =  12412   Num Imagens de Validação =  1551   Num Imagens de Teste =  1552
```

Na segunda parte usei python para fazer a correspondência do código para a “grid search”.

Cinco exemplos de técnicas diferentes de otimização:

- ***Manual Search***: escolhe-se os hiperparametros com base no nosso julgamento, na nossa experiência. Depois treinamos o modelo, avaliamos a sua *accuracy* e repetimos o processo. Paramos, quando o valor da *accuracy* nos é satisfatório;
- ***Random Search***: é onde as diversas combinações de hiperparametros são feitas de forma aleatória e são usadas para encontrar a melhor solução;
- ***Bayesian Optimization***: é uma estratégia de projeto sequencial para funções do tipo “black-box” (é um dispositivo, sistema, ou objeto que produz informações úteis sem revelar nenhuma informação sobre o seu funcionamento interno. Ou seja, as explicações para as suas conclusões permanecem opacas ou “negras”, o que é muito frequente nesta área de Inteligência computacional), a qual reduz também o número de iterações de pesquisa ao escolher os valores de entrada tendo em mente os valores anteriores;
- ***Evolutionary Algorithm***: cria uma população de N modelos de *machine learning* com alguns hiperparametros predefinidos. Depois gera alguns descendentes com hiperparametros semelhantes aos dos melhores modelos para obter novamente uma população de N modelos. Assim, no final do processo, apenas alguns modelos é que vão sobreviver fazendo combinações e variações de parâmetros que são semelhantes à evolução biológica. Este algoritmo simula o processo de seleção natural, o que significa que as espécies que se podem adaptar às mudanças no seu ambiente, podem sobreviver, reproduzir-se e passar para a próxima geração;
- ***Grid Search***: é o método tradicional de otimização de hiperparametros e funciona como uma grelha de hiperparametros e dados de treino e teste em que constrói um modelo para cada combinação de hiperparametros especificados e avalia cada modelo.

6. Análise de resultados

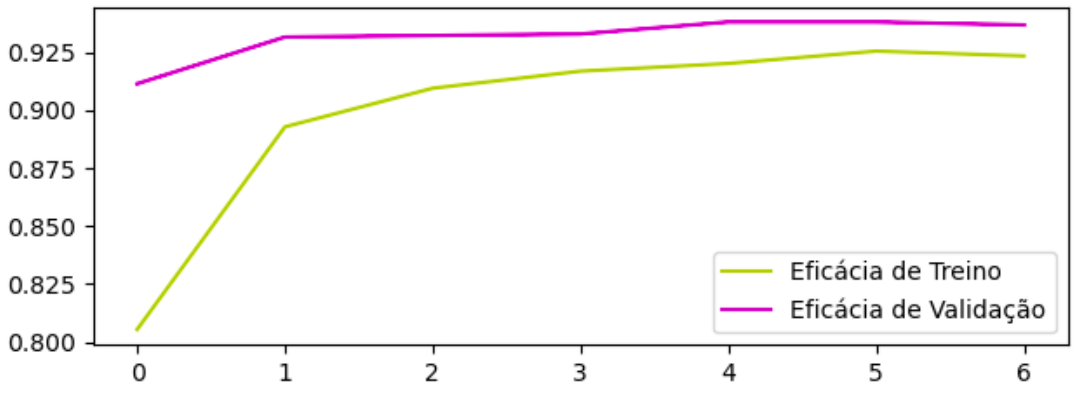
Foram realizados os seguintes testes:

- Camada **GlobalAveragePooling2D** + Função **Softmax** + Data Augmentation;
- Camada **GlobalAveragePooling2D** + Função **Softmax**
- Camada **Flatten** + Função **Softmax**;
- Camada **GlobalAveragePooling2D** + Função **Relu;**
- Camada **Flatten** + Função **Relu**;
- Camada **GlobalAveragePooling2D** + Camada **Flatten** + **Softmax** + Camada **Dropout**.

Camada **GlobalAveragePooling2D** + Função **Softmax** + Data Augmentation:

Foram realizados dois testes com "Augmentation". Este de 9 Epochs e outro de 7, o qual não temos *printscreen*.

```
Epoch 1/20
193/193 [==============================] - 331s 2s/step - loss: 0.6619 - categorical_accuracy: 0.7992 - val_loss:
0.3179 - val_categorical_accuracy: 0.9115
Epoch 2/20
193/193 [==============================] - 341s 2s/step - loss: 0.3374 - categorical_accuracy: 0.8907 - val_loss:
0.2730 - val_categorical_accuracy: 0.9219
Epoch 3/20
193/193 [==============================] - 348s 2s/step - loss: 0.2827 - categorical_accuracy: 0.9060 - val_loss:
0.2535 - val_categorical_accuracy: 0.9219
Epoch 4/20
193/193 [==============================] - 350s 2s/step - loss: 0.2559 - categorical_accuracy: 0.9149 - val_loss:
0.2452 - val_categorical_accuracy: 0.9277
Epoch 5/20
193/193 [==============================] - 342s 2s/step - loss: 0.2430 - categorical_accuracy: 0.9176 - val_loss:
0.2477 - val_categorical_accuracy: 0.9290
Epoch 6/20
193/193 [==============================] - 345s 2s/step - loss: 0.2278 - categorical_accuracy: 0.9266 - val_loss:
0.2529 - val_categorical_accuracy: 0.9277
Epoch 7/20
193/193 [==============================] - 347s 2s/step - loss: 0.2223 - categorical_accuracy: 0.9263 - val_loss:
0.2194 - val_categorical_accuracy: 0.9362
Epoch 8/20
193/193 [==============================] - 348s 2s/step - loss: 0.2137 - categorical_accuracy: 0.9289 - val_loss:
0.2238 - val_categorical_accuracy: 0.9336
Epoch 9/20
193/193 [==============================] - ETA: 0s - loss: 0.2187 - categorical_accuracy: 0.9240Restoring model we
ights from the end of the best epoch: 7.
193/193 [==============================] - 343s 2s/step - loss: 0.2187 - categorical_accuracy: 0.9240 - val_loss:
0.2227 - val_categorical_accuracy: 0.9323
Epoch 9: early stopping
```



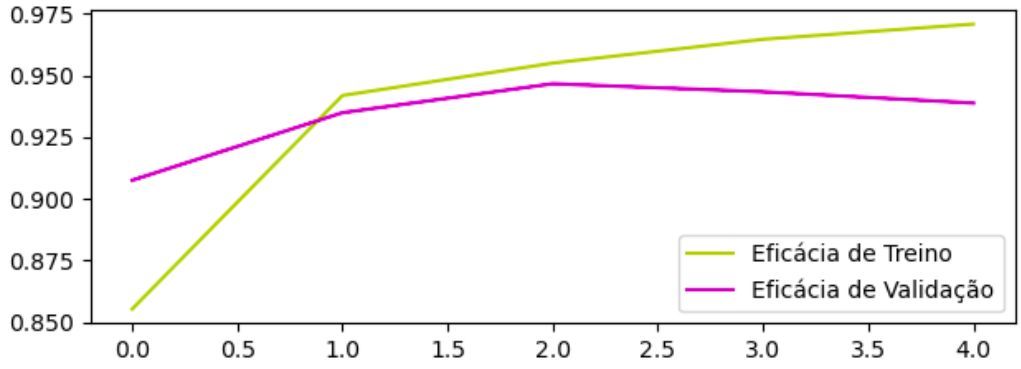
Eficácia de teste = 93.69 %

Camada **GlobalAveragePooling2D** + Função **Softmax**:

Foram realizados dois testes sem Augmentation. Este de 7 Epochs e outro de 5, respetivamente.

```
Epoch 1/20
193/193 [==============================] - 285s 1s/step - loss: 0.4767 - categorical_accuracy: 0.8610 - val_loss:
0.2477 - val_categorical_accuracy: 0.9245
Epoch 2/20
193/193 [==============================] - 291s 2s/step - loss: 0.1956 - categorical_accuracy: 0.9410 - val_loss:
0.2075 - val_categorical_accuracy: 0.9382
Epoch 3/20
193/193 [==============================] - 325s 2s/step - loss: 0.1489 - categorical_accuracy: 0.9563 - val_loss:
0.1813 - val_categorical_accuracy: 0.9434
Epoch 4/20
193/193 [==============================] - 311s 2s/step - loss: 0.1228 - categorical_accuracy: 0.9649 - val_loss:
0.1730 - val_categorical_accuracy: 0.9473
Epoch 5/20
193/193 [==============================] - 298s 2s/step - loss: 0.1027 - categorical_accuracy: 0.9716 - val_loss:
0.1668 - val_categorical_accuracy: 0.9518
Epoch 6/20
193/193 [==============================] - 299s 2s/step - loss: 0.0879 - categorical_accuracy: 0.9761 - val_loss:
0.1620 - val_categorical_accuracy: 0.9512
Epoch 7/20
193/193 [==============================] - ETA: 0s - loss: 0.0778 - categorical_accuracy: 0.9798Restoring model we
ights from the end of the best epoch: 5.
193/193 [==============================] - 300s 2s/step - loss: 0.0778 - categorical_accuracy: 0.9798 - val_loss:
0.1632 - val_categorical_accuracy: 0.9486
Epoch 7: early stopping
```

```
Epoch 1/20
193/193 [==============================] - 284s 1s/step - loss: 0.4882 - categorical_accuracy: 0.8554 - val_loss:
0.2689 - val_categorical_accuracy: 0.9076
Epoch 2/20
193/193 [==============================] - 282s 1s/step - loss: 0.1962 - categorical_accuracy: 0.9419 - val_loss:
0.2049 - val_categorical_accuracy: 0.9349
Epoch 3/20
193/193 [==============================] - 283s 1s/step - loss: 0.1498 - categorical_accuracy: 0.9550 - val_loss:
0.1789 - val_categorical_accuracy: 0.9466
Epoch 4/20
193/193 [==============================] - 279s 1s/step - loss: 0.1230 - categorical_accuracy: 0.9646 - val_loss:
0.1821 - val_categorical_accuracy: 0.9434
Epoch 5/20
193/193 [==============================] - ETA: 0s - loss: 0.1036 - categorical_accuracy: 0.9708Restoring model we
ights from the end of the best epoch: 3.
193/193 [==============================] - 279s 1s/step - loss: 0.1036 - categorical_accuracy: 0.9708 - val_loss:
0.1680 - val_categorical_accuracy: 0.9388
Epoch 5: early stopping
```




```
Eficácia de teste =  94.33 %
```

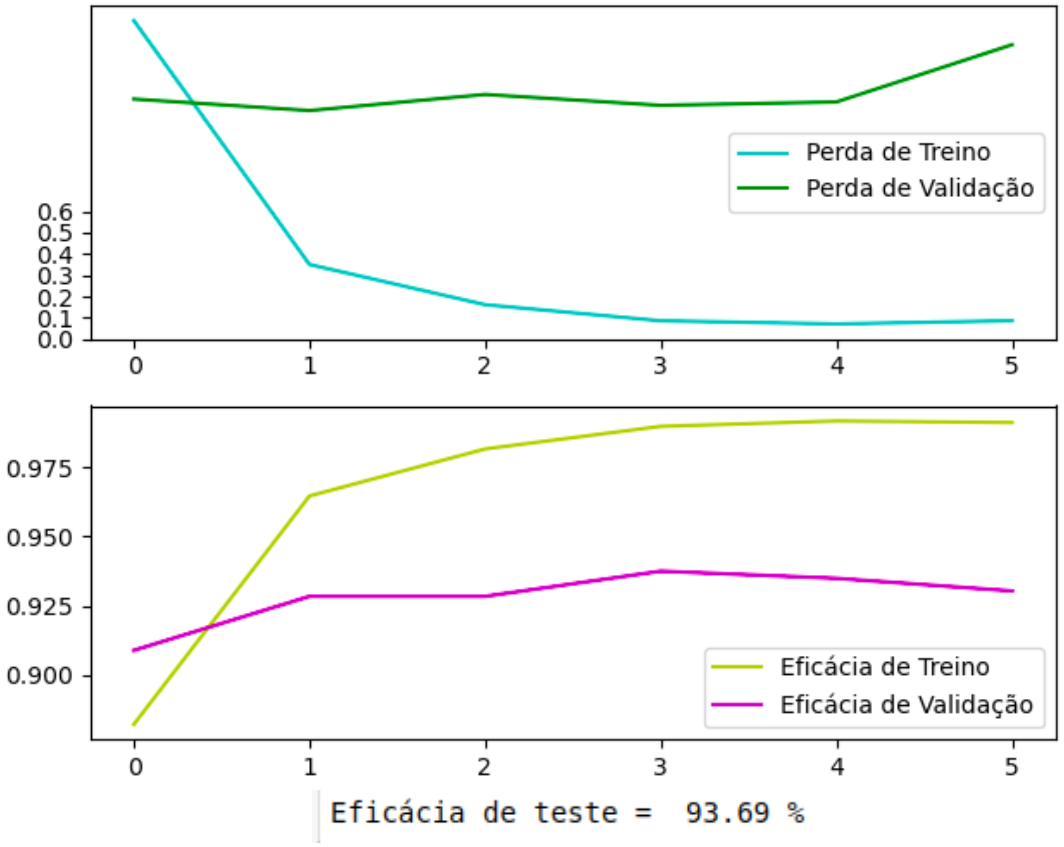
Comparando os resultados mostrados anteriormente, observamos que com "augmentation" conseguimos ter em média mais iterações (epochs), mas, no entanto, a eficácia de teste é maior quando o teste não a tem.

Camada **Flatten** + Função **Softmax**:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 featuresCamadaFlatten (Flat  (None, 62720)            0
 ten)

 dense (Dense)               (None, 10)                627210

=================================================================
Total params: 2,885,194
Trainable params: 627,210
Non-trainable params: 2,257,984
_________________________________________________________________
Número de Features da Camada featuresCamadaFlatten:
tf.Tensor([[0. 0. 0. ... 0. 0. 0.]], shape=(1, 62720), dtype=float32)
```

```
model.add(Flatten(name="featuresCamadaFlatten"))
#model.add(tf.keras.layers.GlobalAveragePooling2D(name="featuresCamadaGlobal"))

model.add(Dense(len(categories), activation='softmax'))
```

```
Epoch 1/20
193/193 [==============================] - 281s 1s/step - loss: 1.4961 - categorical_accuracy: 0.8821 - val_loss:
1.1273 - val_categorical_accuracy: 0.9089
Epoch 2/20
193/193 [==============================] - 285s 1s/step - loss: 0.3504 - categorical_accuracy: 0.9647 - val_loss:
1.0733 - val_categorical_accuracy: 0.9284
Epoch 3/20
193/193 [==============================] - 281s 1s/step - loss: 0.1616 - categorical_accuracy: 0.9817 - val_loss:
1.1490 - val_categorical_accuracy: 0.9284
Epoch 4/20
193/193 [==============================] - 279s 1s/step - loss: 0.0859 - categorical_accuracy: 0.9899 - val_loss:
1.0979 - val_categorical_accuracy: 0.9375
Epoch 5/20
193/193 [==============================] - 279s 1s/step - loss: 0.0712 - categorical_accuracy: 0.9918 - val_loss:
1.1138 - val_categorical_accuracy: 0.9349
Epoch 6/20
193/193 [==============================] - ETA: 0s - loss: 0.0864 - categorical_accuracy: 0.9913Restoring model we
ights from the end of the best epoch: 4.
193/193 [==============================] - 279s 1s/step - loss: 0.0864 - categorical_accuracy: 0.9913 - val_loss:
1.3826 - val_categorical_accuracy: 0.9303
Epoch 6: early stopping
```

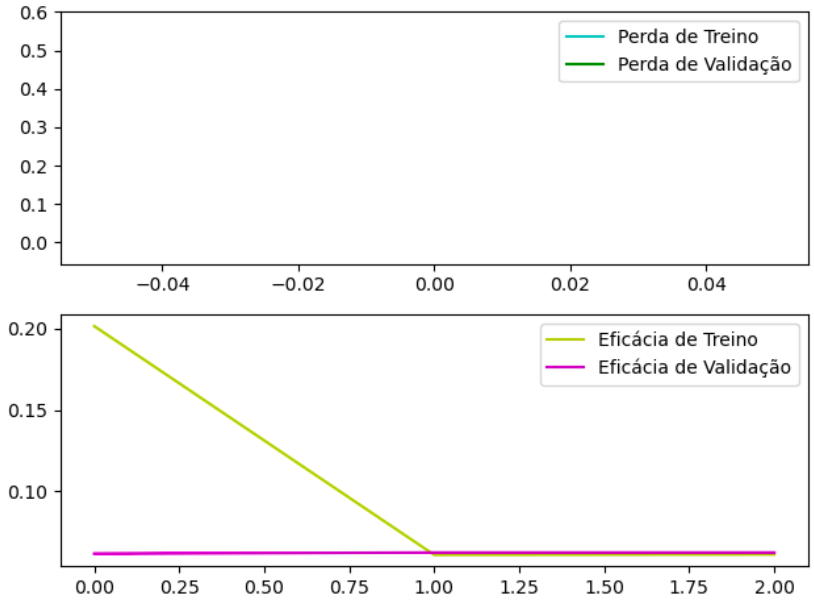



```
Eficácia de teste =  93.69 %
```

Comparando os resultados anteriores, concluímos que a **GlobalAveragePooling2D** tem mais eficácia nos resultados de teste do que a camada **Flatten**.

Camada **GlobalAveragePooling2D** + Função **Relu**:

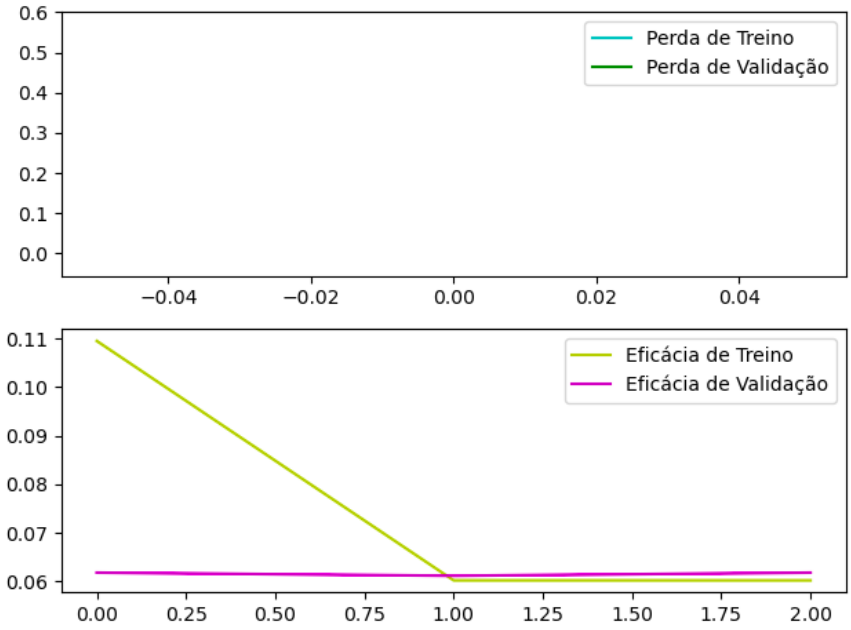
```
Epoch 1/20
193/193 [==============================] - 280s 1s/step - loss: nan - categorical_accuracy: 0.2017 - val_loss: nan
- val_categorical_accuracy: 0.0618
Epoch 2/20
193/193 [==============================] - 280s 1s/step - loss: nan - categorical_accuracy: 0.0611 - val_loss: nan
- val_categorical_accuracy: 0.0625
Epoch 3/20
193/193 [==============================] - ETA: 0s - loss: nan - categorical_accuracy: 0.0613Restoring model weigh
ts from the end of the best epoch: 1.
193/193 [==============================] - 281s 1s/step - loss: nan - categorical_accuracy: 0.0613 - val_loss: nan
- val_categorical_accuracy: 0.0625
Epoch 3: early stopping
```




```
Eficácia de teste =  5.8 %
```

Camada **Flatten** + Função **Relu**;

```
Epoch 1/20
193/193 [==============================] - 290s 1s/step - loss: nan - categorical_accuracy: 0.1095 - val_loss: nan
- val_categorical_accuracy: 0.0618
Epoch 2/20
193/193 [==============================] - 285s 1s/step - loss: nan - categorical_accuracy: 0.0603 - val_loss: nan
- val_categorical_accuracy: 0.0612
Epoch 3/20
193/193 [==============================] - ETA: 0s - loss: nan - categorical_accuracy: 0.0603Restoring model weigh
ts from the end of the best epoch: 1.
193/193 [==============================] - 299s 2s/step - loss: nan - categorical_accuracy: 0.0603 - val_loss: nan
- val_categorical_accuracy: 0.0618
Epoch 3: early stopping
```



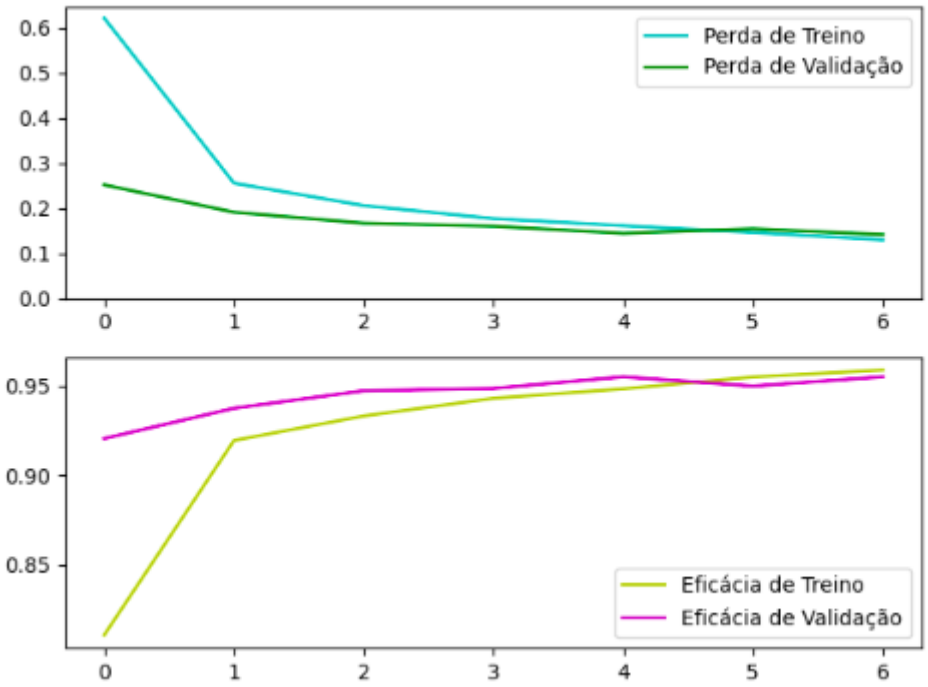

```
Eficácia de teste =  6.51 %
```

Comparando agora estes, observamos que a função de ativação “relu” é sem dúvida inútil para este tipo de classificação.

Camada **GlobalAveragePooling2D** + Camada **Flatten** + **Softmax** + Camada **Dropout**.

```
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 featuresCamadaFlatten (Flat  (None, 1280)             0
 ten)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 10)                12810

=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
_________________________________________________________________
Número de Features da Camada featuresCamadaFlatten:
tf.Tensor([[0.55155    0.         0.44229084 ... 0.         0.4064743  0.12210276]], shape=(1, 1280), dtype=float3
2)
```

```
Epoch 1/20
193/193 [==============================] - 285s 1s/step - loss: 0.6213 - categorical_accuracy: 0.8108 - val_loss:
0.2519 - val_categorical_accuracy: 0.9206
Epoch 2/20
193/193 [==============================] - 279s 1s/step - loss: 0.2557 - categorical_accuracy: 0.9195 - val_loss:
0.1912 - val_categorical_accuracy: 0.9375
Epoch 3/20
193/193 [==============================] - 281s 1s/step - loss: 0.2060 - categorical_accuracy: 0.9332 - val_loss:
0.1669 - val_categorical_accuracy: 0.9473
Epoch 4/20
193/193 [==============================] - 282s 1s/step - loss: 0.1771 - categorical_accuracy: 0.9431 - val_loss:
0.1602 - val_categorical_accuracy: 0.9486
Epoch 5/20
193/193 [==============================] - 282s 1s/step - loss: 0.1614 - categorical_accuracy: 0.9483 - val_loss:
0.1444 - val_categorical_accuracy: 0.9551
Epoch 6/20
193/193 [==============================] - 281s 1s/step - loss: 0.1464 - categorical_accuracy: 0.9550 - val_loss:
0.1546 - val_categorical_accuracy: 0.9499
Epoch 7/20
193/193 [==============================] - ETA: 0s - loss: 0.1301 - categorical_accuracy: 0.9589Restoring model we
ights from the end of the best epoch: 5.
193/193 [==============================] - 282s 1s/step - loss: 0.1301 - categorical_accuracy: 0.9589 - val_loss:
0.1417 - val_categorical_accuracy: 0.9551
Epoch 7: early stopping
```




```
Eficácia de teste =  94.85 %
```

No último teste realizado, conclui-se que a eficácia de treino tende a estar sempre a aumentar, e a eficácia de validação vai acompanhando o treino com uma ligeira perda no epoch 6.

Este foi o teste com maior valor de eficácia – 95%.

Depois de termos a rede neuronal treinada, fomos fazer previsões:

	precision	recall	f1-score	support
battery	0.98	0.93	0.95	85
biological	0.94	0.98	0.96	104
cardboard	0.93	0.98	0.96	87
clothes	0.99	0.98	0.99	539
glass	0.93	0.91	0.92	200
metal	0.87	0.84	0.85	69
paper	0.95	0.93	0.94	114
plastic	0.87	0.87	0.87	99
shoes	0.93	0.97	0.95	191
trash	0.92	0.94	0.93	64
accuracy			0.95	1552
macro avg	0.93	0.93	0.93	1552
weighted avg	0.95	0.95	0.95	1552

Ao olhar para esta imagem percebemos que para as 1552 imagens guardadas para teste, o nosso modelo foi capaz de as reconhecer e de dizer a que categoria é que cada se refere (exemplo - A classe bateria apareceu no conjunto de dados 85 vezes).

7. Conclusões

Com a realização deste trabalho, consegui adquirir bastantes conhecimentos relativamente à temática inteligência computacional e à linguagem de programação *python*.

Com este trabalho foi-nos possível observar que as redes neuronais aparentaram um melhor desempenho quando utilizada a função de treino *softmax*, e as camadas G*lobalAveragePooling2D*, *Flatten* e *Dropout*.

Foi nos também possível aprender que existem diversas técnicas de otimização de hiperparametros com diferentes características entre elas.