ImageClassification-AI: Modelo S

Versão A

Modelo de raiz com o intuito de seguir o que foi lecionado durante o semestre. O objetivo deste modelo, nesta versão, é servir de base e patamar para outros modelos de raiz.

1. Setup

1.1 Importar dependências

Importação das bibliotecas necessárias para o desenvolvimento do modelo.

São de notar as bibliotecas:

- Tensorflow e Keras, que vão ser utilizadas na construção do modelo e no seu processo de treino
- Matplotlib (em específico o pyplot), Seaborn e sklearn, que vão ser utilizadas para facilitar a análise e a compreensão das métricas atribuidas ao modelo, da sua evolução, e dos resultados obtidos
- Image_dataset_from_directory (através do keras.utils), numpy e OS para o carregamento e tratamento dos dados

```
import os
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from keras.utils import image_dataset_from_directory
from tensorflow import keras
from keras import layers, regularizers, optimizers
from sklearn.metrics import confusion_matrix, classification_report
```

1.2 Desativar warnings do Tensorflow

Para desenvolvimento deste modelo foi utilizada a versão 2.10.0 do Tensorflow. Devido a este facto, ficou compreendido que seria beneficial desativar as mensagens de warning dadas pelo Tensorflow, deixando apenas as mensagens de erro, com o intuito de melhorar substancialmente a legibilidade do notebook. É importante realçar que, nenhuma das mensagens de aviso que serão desativadas, em algum momento afetam qualquer aspeto do modelo ou sequer ajudam a compreender potenciais problemas com este.

```
tf.get_logger().setLevel('ERROR')
```

1.3 Tratamento de dados

Definição das classes do problema:

- Tamanho das imagens RGB (32x32x3 pixeis)
- Tamanho de cada batch (32)
- Diretorias dos datasets de treino, validação e teste

Para a criação dos datasets é utilizado o image_dataset_from_directory com os paramêtros relativos à diretoria onde estão as imagens, o tamanho destas, o tamanho de cada batch, a definição das labels como categorical (requerido devido ao facto do problema em questão envolver 10 classes; as labels serão uma tensor float32 de tamanho (batch_size, num_classes), que iram representar, cada, um one-hot encoding de cada index de cada classe).

Aqui é, ainda, importar notar:

- O dataset de treino está a ser baralhado de modo a que, durante o processo de treino, o modelo não decore padrões nas imagens de treino. Para além disso, é relevante perceber que o dataset de treino é construido através da concatenação de quatro datasets de treino mais pequenos (cada um relativo a uma das diretoria de treino)

- Os datasets de validação e de testes não são baralhados. Ao baralhar o dataset de treino a análise dos resultados obtidos pelo modelo seria extremamente dificultada (e.g. ao construir um classification report para este dataset os resultados seriam incorretos porque as labels não iriam corresponder) No que toca ao dataset de validação, a questão entre baralhar ou não acaba por ser irrelevante já que não existe nenhum tipo de benefício para o fazer. Isto foi confirmado por uma pesquisa sobre o assunto e por tentativas de treino do modelo com o dataset de validação baralhado e sem estar baralhado (os resultados eram os mesmo)

```
class_names = []

IMG_SIZE = 32
BATCH_SIZE = 32

train_dirs = ['train1', 'train2', 'train3', 'train5']
val_dir = 'train4'
test_dir = 'test'

print("BUILDING TRAIN DATASET...")
train_dataset_list = []
for td in train_dirs:
    train_dataset_list.append(image_dataset_from_directory(td,
image_size=(IMG_SIZE, IMG_SIZE), batch_size=BATCH_SIZE,
label_mode='categorical', shuffle=True, color_mode='rgb'))

train_dataset = train_dataset_list[0]
for name in train_dataset_list[0].class_names:
    idx = name.index('_') + 1
```

```
    class_names.append(name[idx:])

for d in train_dataset_list[1:]:
    train_dataset = train_dataset.concatenate(d)

print("\nBUILDING VALIDATION DATASET...")
val_dataset = image_dataset_from_directory(val_dir,
image_size=(IMG_SIZE, IMG_SIZE), batch_size=BATCH_SIZE,
label_mode='categorical', shuffle=False,color_mode='rgb')

print("\nBUILDING TEST DATASET...")
test_dataset = image_dataset_from_directory(test_dir,
image_size=(IMG_SIZE, IMG_SIZE), batch_size=BATCH_SIZE,
label_mode='categorical', shuffle=False, color_mode='rgb')
```

```
BUILDING TRAIN DATASET...
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.

BUILDING VALIDATION DATASET...
Found 10000 files belonging to 10 classes.

BUILDING TEST DATASET...
Found 10000 files belonging to 10 classes.
```

2. Visualização

2.1 - Classes e número de imagens

Visualização das classes que envolvem o problema e da quantidade de imagens contidas em cada dataset

```
print("\nClasses: " + str(class_names))

total_train = 0
for td in train_dirs:
    class_folders = next(os.walk(td))[1]
    for cf in class_folders:
        total_train += len(os.listdir(os.path.join(td, cf)))

total_val = 0
class_folders = next(os.walk(val_dir))[1]
for folder in class_folders:
    folder_path = os.path.join(val_dir, folder)
    total_val += len(os.listdir(folder_path))

total_test = 0
class_folders = next(os.walk(test_dir))[1]
```

```
for folder in class_folders:
    folder_path = os.path.join(test_dir, folder)
    total_test += len(os.listdir(folder_path))

print("Dataset de treino: " + str(total_train) + " imagens")
print("Dataset de validação: " + str(total_val) + " imagens")
print("Dataset de teste: " + str(total_test) + " imagens")
```

```
Classes: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']
Dataset de treino: 40000 imagens
Dataset de validação: 10000 imagens
Dataset de teste: 10000 imagens
```

2.2 Tamanhos

Visualização dos tamanhos:

- Cada batch tem 32 imagens
- Cada imagem RGB tem 32x32 pixeis (32x32x3)
- Cada batch de labels tem 10 classes

```
for data_batch, label_batch in train_dataset:
    print('Shape de cada data batch: ', data_batch.shape)
    print('Shape de cada label batch: ', label_batch.shape)
    break
```

```
Shape de cada data batch:  (32, 32, 32, 3)
Shape de cada label batch:  (32, 10)
```

2.3 - Normalização

Visualização da normalização dos pixeis:

- Divisão do valor de cada pixel por 255
- Operação definida, posteriormente, na construção do modelo e, feita durante o processo de treino para cada imagem de modo a que, cada pixel tenha um valor associado que pertença ao intervalo de [0,1].
- Mostrar como o modelo irá interpretar cada imagem (os valores de cada pixel)

```
iterator = train_dataset.as_numpy_iterator()
batch = iterator.next()
batch[0] / 255
```

```
array([[[[0.31764707, 0.32941177, 0.28627452],
         [0.3137255 , 0.3254902 , 0.28235295],
         [0.31764707, 0.32941177, 0.28627452],
         ...,
         [0.34509805, 0.3529412 , 0.3372549 ],
         [0.34117648, 0.34901962, 0.3372549 ],
```

```
         [0.34509805, 0.3529412 , 0.33333334]],

        [[0.30588236, 0.31764707, 0.27450982],
         [0.3019608 , 0.3137255 , 0.27058825],
         [0.3019608 , 0.3137255 , 0.27058825],
         ...,
         [0.34117648, 0.34901962, 0.32941177],
         [0.3372549 , 0.34901962, 0.32156864],
         [0.3372549 , 0.34509805, 0.31764707]],

        [[0.3019608 , 0.3137255 , 0.26666668],
         [0.3019608 , 0.3137255 , 0.27058825],
         [0.30588236, 0.31764707, 0.27450982],
         ...,
         [0.3372549 , 0.34509805, 0.3254902 ],
         [0.33333334, 0.34509805, 0.30588236],
         [0.33333334, 0.34117648, 0.3019608 ]],

        ...,

        [[0.19215687, 0.19607843, 0.13725491],
         [0.1764706 , 0.18039216, 0.12941177],
         [0.14117648, 0.14117648, 0.09803922],
         ...,
         [0.15686275, 0.1764706 , 0.08627451],
         [0.10980392, 0.12941177, 0.05098039],
         [0.10196079, 0.12156863, 0.05490196]],

        [[0.23137255, 0.21568628, 0.14117648],
         [0.20784314, 0.19215687, 0.1254902 ],
         [0.19215687, 0.17254902, 0.11764706],
         ...,
         [0.16078432, 0.18039216, 0.09019608],
         [0.09411765, 0.10980392, 0.04313726],
         [0.08627451, 0.09803922, 0.04705882]],

        [[0.47843137, 0.42745098, 0.32941177],
         [0.4745098 , 0.42352942, 0.3372549 ],
         [0.4745098 , 0.42352942, 0.34117648],
         ...,
         [0.13725491, 0.15294118, 0.07450981],
         [0.08627451, 0.10196079, 0.04313726],
         [0.08235294, 0.09411765, 0.04705882]]],


       [[[0.48235294, 0.5647059 , 0.50980395],
         [0.5254902 , 0.6156863 , 0.5529412 ],
         [0.4745098 , 0.5294118 , 0.48235294],
         ...,
         [0.74509805, 0.8235294 , 0.8156863 ],
```

```
         [0.7254902 , 0.87058824, 0.9098039 ],
         [0.70980394, 0.8627451 , 0.9019608 ]],

        [[0.40392157, 0.44705883, 0.32941177],
         [0.46666667, 0.54901963, 0.45882353],
         [0.40784314, 0.4392157 , 0.38431373],
         ...,
         [0.79607844, 0.5568628 , 0.43137255],
         [0.74509805, 0.7490196 , 0.7058824 ],
         [0.7294118 , 0.8666667 , 0.8901961 ]],

        [[0.34117648, 0.34509805, 0.17254902],
         [0.41960785, 0.47058824, 0.33333334],
         [0.40392157, 0.40784314, 0.3137255 ],
         ...,
         [0.7764706 , 0.45882353, 0.3254902 ],
         [0.7607843 , 0.49411765, 0.36862746],
         [0.7529412 , 0.6627451 , 0.6       ]],

        ...,

        [[0.75686276, 0.64705884, 0.46666667],
         [0.7607843 , 0.654902  , 0.46666667],
         [0.78431374, 0.65882355, 0.4745098 ],
         ...,
         [0.69803923, 0.5529412 , 0.40784314],
         [0.6901961 , 0.56078434, 0.40784314],
         [0.6862745 , 0.5568628 , 0.4117647 ]],

        [[0.75686276, 0.6431373 , 0.45882353],
         [0.77254903, 0.6627451 , 0.47058824],
         [0.7764706 , 0.6509804 , 0.47058824],
         ...,
         [0.7529412 , 0.6431373 , 0.5058824 ],
         [0.69803923, 0.56078434, 0.41568628],
         [0.6901961 , 0.5568628 , 0.41568628]],

        [[0.7490196 , 0.6431373 , 0.4627451 ],
         [0.7647059 , 0.654902  , 0.45882353],
         [0.76862746, 0.6431373 , 0.4509804 ],
         ...,
         [0.7607843 , 0.6509804 , 0.50980395],
         [0.69803923, 0.5647059 , 0.42352942],
         [0.6666667 , 0.53333336, 0.3882353 ]]],


       [[[0.6509804 , 0.5921569 , 0.30588236],
         [0.6862745 , 0.59607846, 0.3019608 ],
         [0.6784314 , 0.627451  , 0.30588236],
         ...,
```

```
         [0.9607843 , 0.9647059 , 0.95686275],
         [0.9764706 , 0.9843137 , 0.9764706 ],
         [0.9764706 , 0.98039216, 0.96862745]],

        [[0.69803923, 0.61960787, 0.28627452],
         [0.7294118 , 0.627451  , 0.2901961 ],
         [0.7254902 , 0.6627451 , 0.29411766],
         ...,
         [0.9764706 , 0.972549  , 0.96862745],
         [1.        , 1.        , 0.99607843],
         [1.        , 0.99607843, 0.9764706 ]],

        [[0.7176471 , 0.62352943, 0.30980393],
         [0.7607843 , 0.63529414, 0.31764707],
         [0.75686276, 0.6745098 , 0.3254902 ],
         ...,
         [0.9529412 , 0.9411765 , 0.85882354],
         [0.9607843 , 0.9490196 , 0.87058824],
         [0.95686275, 0.94509804, 0.85490197]],

        ...,

        [[0.69411767, 0.64705884, 0.38431373],
         [0.73333335, 0.6627451 , 0.39607844],
         [0.6784314 , 0.61960787, 0.34509805],
         ...,
         [0.8235294 , 0.7176471 , 0.4627451 ],
         [0.81960785, 0.7176471 , 0.45490196],
         [0.8117647 , 0.7058824 , 0.44705883]],

        [[0.7137255 , 0.61960787, 0.30980393],
         [0.7529412 , 0.63529414, 0.32941177],
         [0.68235296, 0.5764706 , 0.26666668],
         ...,
         [0.81960785, 0.7019608 , 0.43529412],
         [0.8117647 , 0.69411767, 0.41960785],
         [0.8       , 0.6901961 , 0.41960785]],

        [[0.6666667 , 0.6       , 0.32156864],
         [0.7058824 , 0.6117647 , 0.34117648],
         [0.627451  , 0.54509807, 0.27450982],
         ...,
         [0.8117647 , 0.6862745 , 0.4117647 ],
         [0.7882353 , 0.6627451 , 0.38039216],
         [0.77254903, 0.67058825, 0.4       ]]],


       ...,
```

```
       [[[0.7411765 , 0.7372549 , 0.7294118 ],
         [0.7019608 , 0.7137255 , 0.7176471 ],
         [0.6627451 , 0.69803923, 0.7058824 ],
         ...,
         [0.64705884, 0.7019608 , 0.7137255 ],
         [0.6431373 , 0.7058824 , 0.7137255 ],
         [0.6431373 , 0.7019608 , 0.7176471 ]],

        [[0.7254902 , 0.73333335, 0.7372549 ],
         [0.69411767, 0.7176471 , 0.7254902 ],
         [0.6666667 , 0.70980394, 0.7176471 ],
         ...,
         [0.6666667 , 0.7058824 , 0.7254902 ],
         [0.6627451 , 0.70980394, 0.7254902 ],
         [0.65882355, 0.70980394, 0.7254902 ]],

        [[0.6509804 , 0.68235296, 0.69411767],
         [0.6431373 , 0.6862745 , 0.69803923],
         [0.6392157 , 0.69411767, 0.7019608 ],
         ...,
         [0.6666667 , 0.69803923, 0.7176471 ],
         [0.6627451 , 0.7019608 , 0.7176471 ],
         [0.654902  , 0.7058824 , 0.72156864]],

        ...,

        [[0.48235294, 0.53333336, 0.5568628 ],
         [0.47843137, 0.5294118 , 0.5529412 ],
         [0.48235294, 0.53333336, 0.5568628 ],
         ...,
         [0.48235294, 0.53333336, 0.5568628 ],
         [0.48235294, 0.53333336, 0.5568628 ],
         [0.48235294, 0.53333336, 0.5568628 ]],

        [[0.4862745 , 0.5372549 , 0.56078434],
         [0.48235294, 0.53333336, 0.5568628 ],
         [0.4862745 , 0.5372549 , 0.56078434],
         ...,
         [0.4862745 , 0.5372549 , 0.56078434],
         [0.4862745 , 0.5372549 , 0.56078434],
         [0.4862745 , 0.5372549 , 0.56078434]],

        [[0.47843137, 0.5294118 , 0.5529412 ],
         [0.47843137, 0.5254902 , 0.54901963],
         [0.47843137, 0.5294118 , 0.5529412 ],
         ...,
         [0.47843137, 0.5294118 , 0.5529412 ],
         [0.47843137, 0.5294118 , 0.5529412 ],
         [0.4745098 , 0.5254902 , 0.54901963]]],
```

```
       [[[0.5568628 , 0.5568628 , 0.5137255 ],
         [0.6666667 , 0.6627451 , 0.6313726 ],
         [0.6745098 , 0.67058825, 0.6509804 ],
         ...,
         [0.40392157, 0.3882353 , 0.38039216],
         [0.41568628, 0.40392157, 0.4       ],
         [0.7411765 , 0.73333335, 0.7372549 ]],

        [[0.4862745 , 0.4862745 , 0.4627451 ],
         [0.5568628 , 0.5529412 , 0.5372549 ],
         [0.54509807, 0.5411765 , 0.5294118 ],
         ...,
         [0.43137255, 0.41568628, 0.39607844],
         [0.34901962, 0.34117648, 0.3254902 ],
         [0.7176471 , 0.7137255 , 0.7019608 ]],

        [[0.4509804 , 0.44313726, 0.44313726],
         [0.45882353, 0.4509804 , 0.45490196],
         [0.5529412 , 0.54509807, 0.5529412 ],
         ...,
         [0.6       , 0.5882353 , 0.54901963],
         [0.5764706 , 0.5686275 , 0.5372549 ],
         [0.6313726 , 0.627451  , 0.59607846]],

        ...,

        [[0.6745098 , 0.627451  , 0.6039216 ],
         [0.6862745 , 0.6392157 , 0.6117647 ],
         [0.6392157 , 0.59607846, 0.5647059 ],
         ...,
         [0.7411765 , 0.7137255 , 0.6784314 ],
         [0.6509804 , 0.62352943, 0.58431375],
         [0.69803923, 0.67058825, 0.6313726 ]],

        [[0.68235296, 0.6117647 , 0.5882353 ],
         [0.7176471 , 0.6392157 , 0.6       ],
         [0.7607843 , 0.6901961 , 0.6431373 ],
         ...,
         [0.73333335, 0.7058824 , 0.6745098 ],
         [0.6784314 , 0.6509804 , 0.6156863 ],
         [0.5647059 , 0.5411765 , 0.5019608 ]],

        [[0.69803923, 0.627451  , 0.6039216 ],
         [0.7176471 , 0.6392157 , 0.6       ],
         [0.7647059 , 0.69411767, 0.64705884],
         ...,
         [0.7019608 , 0.6745098 , 0.6431373 ],
         [0.62352943, 0.59607846, 0.56078434],
         [0.5254902 , 0.49803922, 0.45882353]]],
```

```
       [[[0.29803923, 0.50980395, 0.47058824],
         [0.32156864, 0.54901963, 0.5019608 ],
         [0.30980393, 0.5529412 , 0.5019608 ],
         ...,
         [0.3254902 , 0.6039216 , 0.54901963],
         [0.3019608 , 0.5764706 , 0.5176471 ],
         [0.35686275, 0.627451  , 0.5686275 ]],

        [[0.20392157, 0.3764706 , 0.34901962],
         [0.23921569, 0.45490196, 0.4117647 ],
         [0.3254902 , 0.5803922 , 0.5254902 ],
         ...,
         [0.36078432, 0.627451  , 0.5764706 ],
         [0.28627452, 0.54509807, 0.49411765],
         [0.30980393, 0.5764706 , 0.5176471 ]],

        [[0.2       , 0.34901962, 0.32941177],
         [0.19607843, 0.39215687, 0.35686275],
         [0.2509804 , 0.49803922, 0.44705883],
         ...,
         [0.37254903, 0.62352943, 0.5803922 ],
         [0.3764706 , 0.627451  , 0.5764706 ],
         [0.33333334, 0.6       , 0.5411765 ]],

        ...,

        [[0.2       , 0.3764706 , 0.34901962],
         [0.2       , 0.3764706 , 0.34901962],
         [0.2       , 0.3764706 , 0.34901962],
         ...,
         [0.23921569, 0.44313726, 0.40784314],
         [0.25490198, 0.4509804 , 0.41568628],
         [0.2509804 , 0.4509804 , 0.41568628]],

        [[0.27058825, 0.45490196, 0.42352942],
         [0.23137255, 0.4117647 , 0.38431373],
         [0.2       , 0.3764706 , 0.34901962],
         ...,
         [0.21176471, 0.35686275, 0.34117648],
         [0.14117648, 0.29411766, 0.27450982],
         [0.20392157, 0.3882353 , 0.35686275]],

        [[0.26666668, 0.42352942, 0.4       ],
         [0.2509804 , 0.4       , 0.38039216],
         [0.23921569, 0.38039216, 0.36078432],
         ...,
         [0.13725491, 0.21960784, 0.21568628],
```

```
        [0.09019608, 0.19607843, 0.1882353 ],
        [0.21960784, 0.36862746, 0.34509805]]]], dtype=float32)
```

2.4 - Imagens do dataset de treino

Visualização de dez imagens aleatórias do dataset de treino

```
plt.figure(figsize=(12, 6))  # Aumentar o tamanho das imagens no plot

for data_batch, label_batch in train_dataset.take(1):
    for i in range(10):
        plt.subplot(2, 5, i + 1) # mostrar as imagens todas no "mesmo plot" de modo a economizar espaço
        plt.title(class_names[np.argmax(label_batch[i])]) # mostrar a classe da imagem
        plt.imshow(data_batch[i].numpy().astype('uint8'))
        plt.xticks([]) # não mostrar os eixos (irrelevante para a visualização)
        plt.yticks([]) # não mostrar os eixos (irrelevante para a visualização)
    plt.show()
```

3. Modelo

3.1 Definição

Apesar deste modelo ter o objetivo de seguir aquilo que foi lecionado ao longo do semestre, é necessário alterar a arquitetura do modelo treinado durante as aulas devido ao facto de estarem a ser utilizadas imagens mais pequenas. Caso isso não fosse precavido, os feature maps criados pelo modelo iriam ter tamanhos inválidos.

Na arquitetura deste modelo temos:

- Como supramencionado, a normalização dos valores de cada pixel da imagem
- Três blocos de layers convolucionais:
  - Cada um com uma layer convulocional
  - A quantidade de filtros em cada camada vai aumentado progressivamente de 32 filtros até 128
  - É utilizada a função de ativação ReLu
  - No final de cada bloco é feito o MaxPooling do feature map até aquele momento, com um filtro de 2x2 (que irá reduzir o tamanho de feature map em metade e, no caso de o valor ser decimal, irá arrendondar o tamanho às unidades)
  - É utilizada a técnica de regularização BatchNormalization com o intuito de manter consistente a distribuição dos valores que saem dos outputs de cada layer e que entram na próxima
- Bloco de classificação:
  - É utilizado o Flatten para transformar os valores obtidos até aqui num vetor 1D
  - É utilizada uma camada densa com 128 filtros que, irá receber os valores da ultima camada convolucional aos quais vai aplicar a BatchNormalization e a técnica de Dropout (que consiste em excluir, aleatóriamente, x * 100 % dos neurónios anteriores, sendo x o valor que passamos por paramêtro). O Dropout é utilizado especialmente para combater o overfitting.
  - É utilizada uma outra camada densa, com 10 filtros (relativos à quantidade de classes do problema), para efetuar a classificação da imagem. Aqui é utilizada a função de ativação "softmax" devido a esta ser mais apropriada a um problema de classificação com várias classes diferentes. Para além disso, é também, utilizado a regularização L2 para, tal como o Dropout, combater o overfitting

É feito um sumário do modelo para melhor compreensão deste, especialmente no que toca ao tamanho dos feature maps em cada ponto e à quantidade de paramêtros que este envolve.

```
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

x = layers.Rescaling(1./255)(inputs)

# 1st Convolutional Block/Layer
x = layers.Conv2D(filters=32, kernel_size=3)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

# 2nd Convolutional Block/Layer
x = layers.Conv2D(filters=64, kernel_size=3)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
# 3rd Convolutional Block/Layer
x = layers.Conv2D(filters=128, kernel_size=3)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Flatten()(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(10, activation="softmax",
kernel_regularizer=regularizers.l2(0.01))(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.summary()
```

```
Model: "functional_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ input_layer (InputLayer)             │ (None, 32, 32, 3)           │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ rescaling (Rescaling)                │ (None, 32, 32, 3)           │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d (Conv2D)                      │ (None, 30, 30, 32)          │
896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization                  │ (None, 30, 30, 32)          │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ activation (Activation)              │ (None, 30, 30, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d (MaxPooling2D)         │ (None, 15, 15, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
```

```
──────┤
| conv2d_1 (Conv2D)                | (None, 13, 13, 64)      |
18,496 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| batch_normalization_1            | (None, 13, 13, 64)      |
256 |
| (BatchNormalization)             |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| activation_1 (Activation)        | (None, 13, 13, 64)      |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| max_pooling2d_1 (MaxPooling2D)   | (None, 6, 6, 64)        |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_2 (Conv2D)                | (None, 4, 4, 128)       |
73,856 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| batch_normalization_2            | (None, 4, 4, 128)       |
512 |
| (BatchNormalization)             |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| activation_2 (Activation)        | (None, 4, 4, 128)       |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| max_pooling2d_2 (MaxPooling2D)   | (None, 2, 2, 128)       |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| flatten (Flatten)                | (None, 512)             |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| dense (Dense)                    | (None, 128)             |
65,664 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| batch_normalization_3            | (None, 128)             |
512 |
| (BatchNormalization)             |                         |
|
```

```
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| dropout (Dropout)                 | (None, 128)                 |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| dense_1 (Dense)                   | (None, 10)                  |
1,290 |
└─────────────────────────────────────┴─────────────────────────────┴──────────
──────┘

 Total params: 161,610 (631.29 KB)

 Trainable params: 160,906 (628.54 KB)

 Non-trainable params: 704 (2.75 KB)
```

3.2 Compilação

É utilizada a função de loss "categorical_crossentropy" devido à natureza do problema (várias classes). Para analisar o desempenho do modelo são utilizadas metricas de acerto (neste caso o "CategorialAccuracy" em vez do Accuracy normal devido ao contexto do problema), precisão e recall. É, ainda, importante referir que inicialmente era para ser incluida uma métrica de calculo relativo ao F1-Score, mas, devido ao facto de ter sido utilizado o Tensorflow 2.10.0 para treinar os modelos, como supramencionado, não foi possível utilizar esta métrica. Isto acontece porque esta versão do Tensorflow não suporta a referida metrica. Realizaram-se experiências utilizando a métrica F1-Score do Tensorflow Addons mas, os resultados não foram satisfatórios.

Como este modelo visa se aproximar ao máximo, dentro dos possíveis, aos exemplos lecionados ao longo do semestre foi escolhido o optimizador RMSprop com um learning rate de 0.0001.

```
model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizers.RMSprop(learning_rate=1e-4),
    metrics=[
                tf.keras.metrics.CategoricalAccuracy(name='accuracy'),
                tf.keras.metrics.Precision(name='precision'),
                tf.keras.metrics.Recall(name='recall'),
            ])
```

3.3 Processo de treino

São definidas callbacks de:

- EarlyStopping, que vai servir para interromper o processo de treino. É monitorizada a loss no dataset de validação em cada epoch e, se após 10 epochs não houver melhoria desta métrica, então o treino vai ser interrompido
- ModelCheckpoint, que vai permitir guardar o melhor modelo obtido durante o processo de treino (em troca de se guardar o modelo na ultima epoch de treino que,

pode não ser necessáriamente o melhor como é o caso de, por exemplo, situações onde o modelo começa a entrar em overfitting). Aqui é definida a diretoria onde guardar o melhor modelo e a metrica de monitorização que, neste caso, volta a ser a loss no dataset de validação. É, também utilizado o verbose para melhorar a compreensão do processo de treino.

Com isto, é, então, realizado o processo de treino (model.fit) utilzando:

- O dataset de treino
- 100 epochs
- O dataset de validação para representar a capacidade de generalização do modelo
- As callbacks de EarlyStopping e ModelCheckpoint definidas

```
# Definir as callbacks
callbacks = [
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=10,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath='models/IC_S_A.keras',
        save_best_only = True,
        monitor='val_loss',
        verbose=1
    )
]

# Treinar o modelo
history = model.fit(train_dataset, epochs=100,
validation_data=val_dataset, callbacks=callbacks)
```

```
Epoch 1/100

1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.2564 - loss:
2.6119 - precision: 0.3251 - recall: 0.1423
Epoch 1: val_loss improved from inf to 1.57731, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 88s 66ms/step - accuracy: 0.2565 -
loss: 2.6114 - precision: 0.3252 - recall: 0.1424 - val_accuracy:
0.4801 - val_loss: 1.5773 - val_precision: 0.6627 - val_recall: 0.2688
Epoch 2/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.4382 - loss:
1.7448 - precision: 0.5567 - recall: 0.2952
Epoch 2: val_loss improved from 1.57731 to 1.39191, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 49s 39ms/step - accuracy: 0.4382 -
loss: 1.7447 - precision: 0.5567 - recall: 0.2953 - val_accuracy:
0.5315 - val_loss: 1.3919 - val_precision: 0.7061 - val_recall: 0.3512
Epoch 3/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.5195 - loss:
```

```
1.4684 - precision: 0.6526 - recall: 0.3714
Epoch 3: val_loss improved from 1.39191 to 1.29600, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 53s 42ms/step - accuracy: 0.5195 -
loss: 1.4683 - precision: 0.6526 - recall: 0.3715 - val_accuracy:
0.5727 - val_loss: 1.2960 - val_precision: 0.7424 - val_recall: 0.4008
Epoch 4/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.5703 - loss:
1.3092 - precision: 0.7079 - recall: 0.4214
Epoch 4: val_loss improved from 1.29600 to 1.21375, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 35s 28ms/step - accuracy: 0.5703 -
loss: 1.3091 - precision: 0.7079 - recall: 0.4214 - val_accuracy:
0.5935 - val_loss: 1.2138 - val_precision: 0.7394 - val_recall: 0.4505
Epoch 5/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6074 - loss:
1.2006 - precision: 0.7424 - recall: 0.4569
Epoch 5: val_loss improved from 1.21375 to 1.12818, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.6074 -
loss: 1.2005 - precision: 0.7424 - recall: 0.4569 - val_accuracy:
0.6320 - val_loss: 1.1282 - val_precision: 0.7789 - val_recall: 0.4752
Epoch 6/100
1249/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6300 - loss:
1.1287 - precision: 0.7667 - recall: 0.4858
Epoch 6: val_loss improved from 1.12818 to 1.10700, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 31s 25ms/step - accuracy: 0.6300 -
loss: 1.1286 - precision: 0.7667 - recall: 0.4859 - val_accuracy:
0.6371 - val_loss: 1.1070 - val_precision: 0.7740 - val_recall: 0.4952
Epoch 7/100
1249/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6504 - loss:
1.0696 - precision: 0.7785 - recall: 0.5108
Epoch 7: val_loss improved from 1.10700 to 1.08232, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 32s 25ms/step - accuracy: 0.6505 -
loss: 1.0695 - precision: 0.7785 - recall: 0.5108 - val_accuracy:
0.6452 - val_loss: 1.0823 - val_precision: 0.7656 - val_recall: 0.5103
Epoch 8/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6712 - loss:
1.0114 - precision: 0.8010 - recall: 0.5407
Epoch 8: val_loss improved from 1.08232 to 1.06356, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 29s 23ms/step - accuracy: 0.6712 -
loss: 1.0114 - precision: 0.8010 - recall: 0.5407 - val_accuracy:
0.6487 - val_loss: 1.0636 - val_precision: 0.7679 - val_recall: 0.5327
Epoch 9/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6849 - loss:
0.9680 - precision: 0.8087 - recall: 0.5557
```

```
Epoch 9: val_loss improved from 1.06356 to 1.01123, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 31s 25ms/step - accuracy: 0.6849 -
loss: 0.9679 - precision: 0.8087 - recall: 0.5558 - val_accuracy:
0.6657 - val_loss: 1.0112 - val_precision: 0.7879 - val_recall: 0.5450
Epoch 10/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7040 - loss:
0.9238 - precision: 0.8213 - recall: 0.5829
Epoch 10: val_loss improved from 1.01123 to 0.96747, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 34s 27ms/step - accuracy: 0.7040 -
loss: 0.9238 - precision: 0.8213 - recall: 0.5829 - val_accuracy:
0.6820 - val_loss: 0.9675 - val_precision: 0.7993 - val_recall: 0.5734
Epoch 11/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7137 - loss:
0.8887 - precision: 0.8229 - recall: 0.5973
Epoch 11: val_loss improved from 0.96747 to 0.96048, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.7137 -
loss: 0.8887 - precision: 0.8230 - recall: 0.5973 - val_accuracy:
0.6832 - val_loss: 0.9605 - val_precision: 0.8017 - val_recall: 0.5781
Epoch 12/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7265 - loss:
0.8528 - precision: 0.8319 - recall: 0.6148
Epoch 12: val_loss did not improve from 0.96048
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 34s 27ms/step - accuracy: 0.7265 -
loss: 0.8528 - precision: 0.8319 - recall: 0.6148 - val_accuracy:
0.6787 - val_loss: 0.9739 - val_precision: 0.7834 - val_recall: 0.5815
Epoch 13/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7419 - loss:
0.8187 - precision: 0.8396 - recall: 0.6344
Epoch 13: val_loss did not improve from 0.96048
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 32s 25ms/step - accuracy: 0.7419 -
loss: 0.8187 - precision: 0.8396 - recall: 0.6344 - val_accuracy:
0.6790 - val_loss: 0.9860 - val_precision: 0.7743 - val_recall: 0.5874
Epoch 14/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7485 - loss:
0.7904 - precision: 0.8421 - recall: 0.6538
Epoch 14: val_loss improved from 0.96048 to 0.94112, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.7485 -
loss: 0.7904 - precision: 0.8421 - recall: 0.6538 - val_accuracy:
0.6915 - val_loss: 0.9411 - val_precision: 0.7929 - val_recall: 0.6060
Epoch 15/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.7641 - loss:
0.7593 - precision: 0.8529 - recall: 0.6661
Epoch 15: val_loss did not improve from 0.94112
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 34s 27ms/step - accuracy: 0.7641 -
loss: 0.7593 - precision: 0.8529 - recall: 0.6661 - val_accuracy:
```

```
0.6871 - val_loss: 0.9583 - val_precision: 0.7847 - val_recall: 0.6025
Epoch 16/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7669 - loss:
0.7351 - precision: 0.8525 - recall: 0.6818
Epoch 16: val_loss did not improve from 0.94112
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 32s 26ms/step - accuracy: 0.7669 -
loss: 0.7351 - precision: 0.8525 - recall: 0.6818 - val_accuracy:
0.6929 - val_loss: 0.9445 - val_precision: 0.7832 - val_recall: 0.6095
Epoch 17/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7788 - loss:
0.7059 - precision: 0.8613 - recall: 0.6951
Epoch 17: val_loss improved from 0.94112 to 0.93243, saving model to
models/IC_S_A.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.7788 -
loss: 0.7059 - precision: 0.8613 - recall: 0.6951 - val_accuracy:
0.6967 - val_loss: 0.9324 - val_precision: 0.7818 - val_recall: 0.6233
Epoch 18/100
1249/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7863 - loss:
0.6825 - precision: 0.8648 - recall: 0.7043
Epoch 18: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 36s 29ms/step - accuracy: 0.7863 -
loss: 0.6825 - precision: 0.8648 - recall: 0.7043 - val_accuracy:
0.6815 - val_loss: 0.9823 - val_precision: 0.7650 - val_recall: 0.6080
Epoch 19/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7963 - loss:
0.6606 - precision: 0.8689 - recall: 0.7190
Epoch 19: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 32s 26ms/step - accuracy: 0.7963 -
loss: 0.6606 - precision: 0.8689 - recall: 0.7190 - val_accuracy:
0.6744 - val_loss: 1.0068 - val_precision: 0.7558 - val_recall: 0.6055
Epoch 20/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8036 - loss:
0.6437 - precision: 0.8740 - recall: 0.7325
Epoch 20: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 36s 29ms/step - accuracy: 0.8036 -
loss: 0.6437 - precision: 0.8740 - recall: 0.7325 - val_accuracy:
0.6909 - val_loss: 0.9686 - val_precision: 0.7643 - val_recall: 0.6314
Epoch 21/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8110 - loss:
0.6186 - precision: 0.8779 - recall: 0.7431
Epoch 21: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.8110 -
loss: 0.6186 - precision: 0.8779 - recall: 0.7431 - val_accuracy:
0.6754 - val_loss: 1.0417 - val_precision: 0.7478 - val_recall: 0.6218
Epoch 22/100
1250/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8180 - loss:
0.5987 - precision: 0.8813 - recall: 0.7523
Epoch 22: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.8180 -
```

```
loss: 0.5987 - precision: 0.8813 - recall: 0.7523 - val_accuracy:
0.6834 - val_loss: 1.0151 - val_precision: 0.7514 - val_recall: 0.6278
Epoch 23/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8247 - loss:
0.5762 - precision: 0.8849 - recall: 0.7590
Epoch 23: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.8247 -
loss: 0.5762 - precision: 0.8849 - recall: 0.7590 - val_accuracy:
0.6880 - val_loss: 0.9882 - val_precision: 0.7548 - val_recall: 0.6285
Epoch 24/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8327 - loss:
0.5554 - precision: 0.8890 - recall: 0.7712
Epoch 24: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 34s 27ms/step - accuracy: 0.8327 -
loss: 0.5554 - precision: 0.8890 - recall: 0.7712 - val_accuracy:
0.6833 - val_loss: 1.0177 - val_precision: 0.7483 - val_recall: 0.6328
Epoch 25/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8421 - loss:
0.5349 - precision: 0.8957 - recall: 0.7857
Epoch 25: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 32s 25ms/step - accuracy: 0.8421 -
loss: 0.5348 - precision: 0.8957 - recall: 0.7857 - val_accuracy:
0.6813 - val_loss: 1.0372 - val_precision: 0.7415 - val_recall: 0.6303
Epoch 26/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8451 - loss:
0.5196 - precision: 0.8977 - recall: 0.7905
Epoch 26: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 27ms/step - accuracy: 0.8451 -
loss: 0.5196 - precision: 0.8977 - recall: 0.7905 - val_accuracy:
0.6594 - val_loss: 1.1269 - val_precision: 0.7187 - val_recall: 0.6110
Epoch 27/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8469 - loss:
0.5078 - precision: 0.8978 - recall: 0.7948
Epoch 27: val_loss did not improve from 0.93243
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 33s 26ms/step - accuracy: 0.8469 -
loss: 0.5078 - precision: 0.8978 - recall: 0.7948 - val_accuracy:
0.6889 - val_loss: 1.0290 - val_precision: 0.7453 - val_recall: 0.6478
```

3.4 Avaliação

O melhor modelo obtido durante o processo de treino é carregado e avaliado utilizando o dataset de teste. Aqui são mostrados os valores das métricas de accuracy, loss, precision e recall obtidas pelo modelo nas imagens de teste.

```
# Carregar o modelo
model = keras.models.load_model('models/IC_S_A.keras')

# Avaliar o modelo utilizando o dataset de testes
test_loss, test_acc, test_precision, test_recall =
```

```
model.evaluate(test_dataset)

print("Test Accuracy: " + str(test_acc))
print("Test Loss: " + str(test_loss))
print("Test Precision: " + str(test_precision))
print("Test Recall: " + str(test_recall))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.6923 - loss:
0.9583 - precision: 0.7639 - recall: 0.6133
Test Accuracy: 0.703499972820282
Test Loss: 0.9201842546463013
Test Precision: 0.7827009558677673
Test Recall: 0.6270999908447266
```

4. Análise de resultados

4.1 Evolução das métricas durante o processo de treino

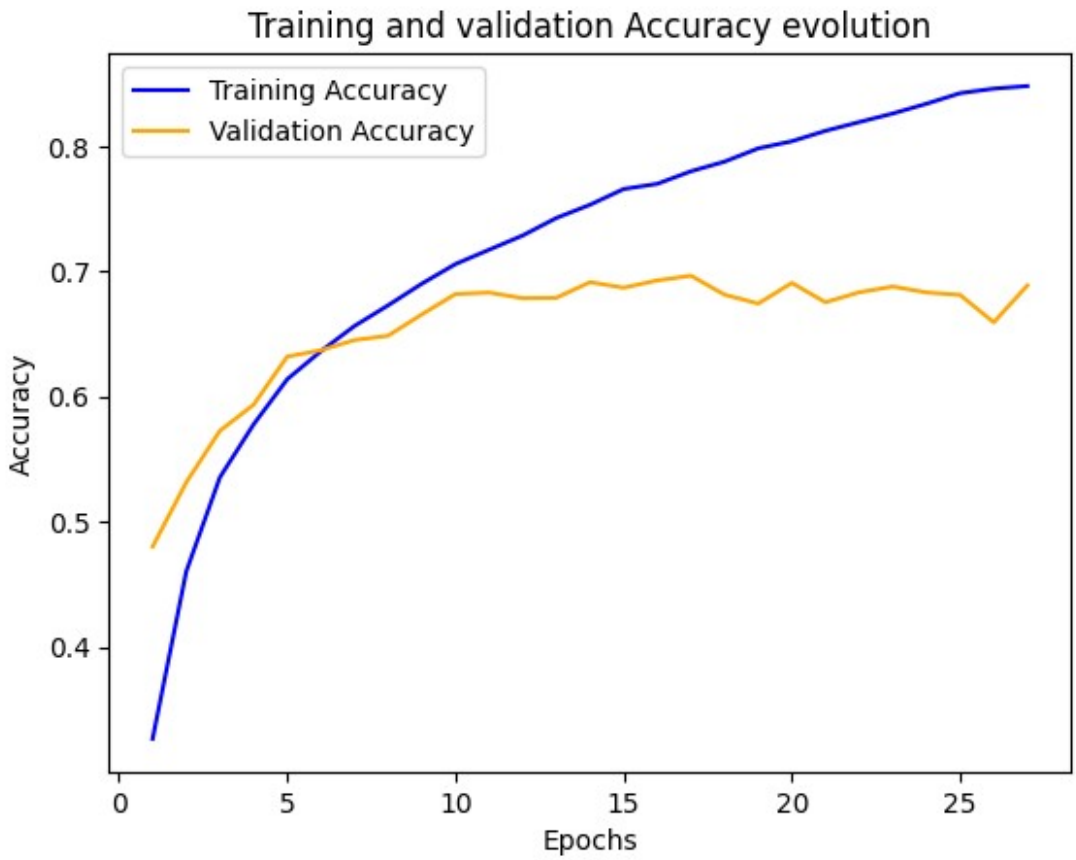
São utilizados gráficos para melhor compreender de que maneira as métricas, nomeadamente a accuracy, loss, precision e recall, foram evoluindo ao longo do processo de treino.

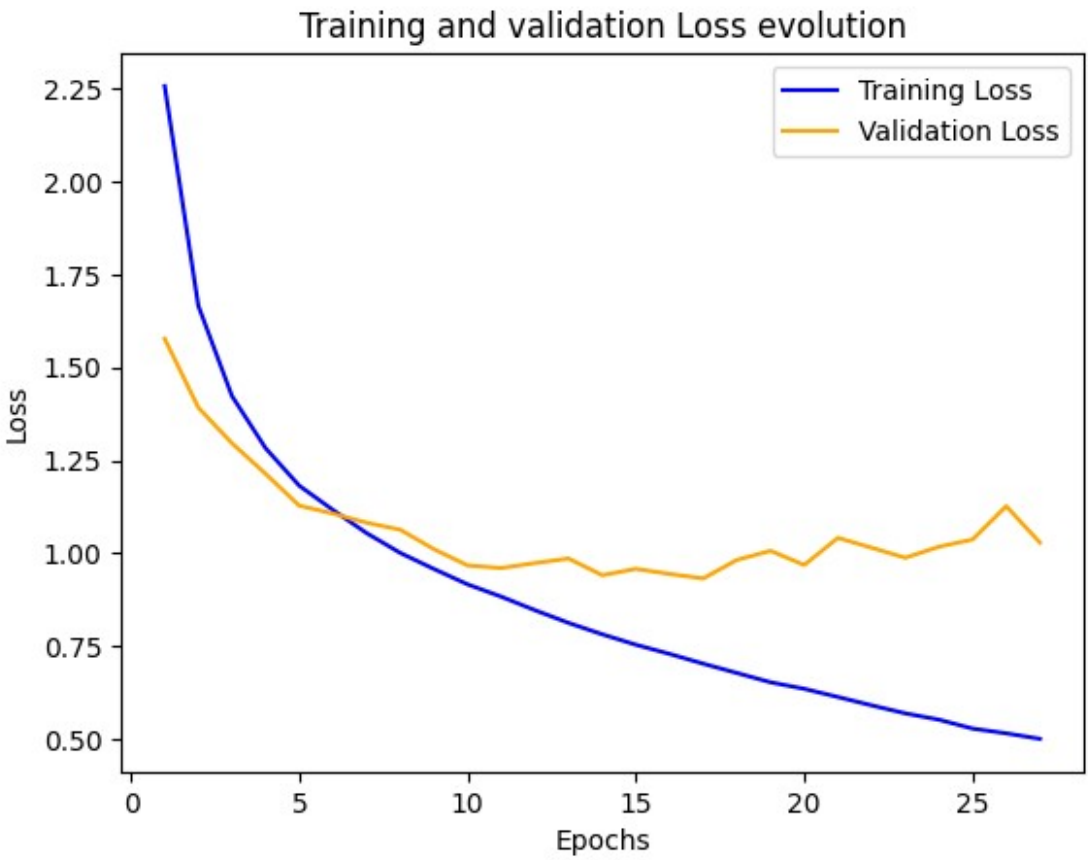
É possível visualizar que:

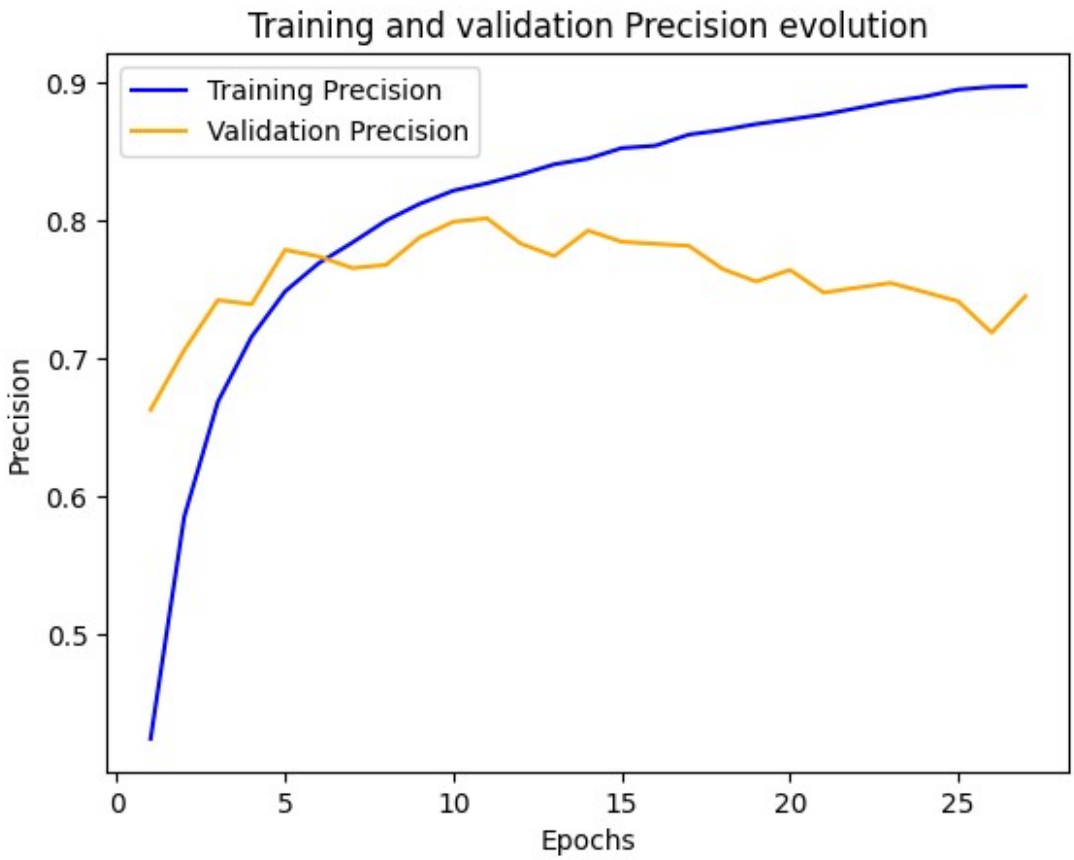
- Por volta da época 16 o modelo começa a entrar em overfitting (o melhor modelo obtido e guardado é desta época)

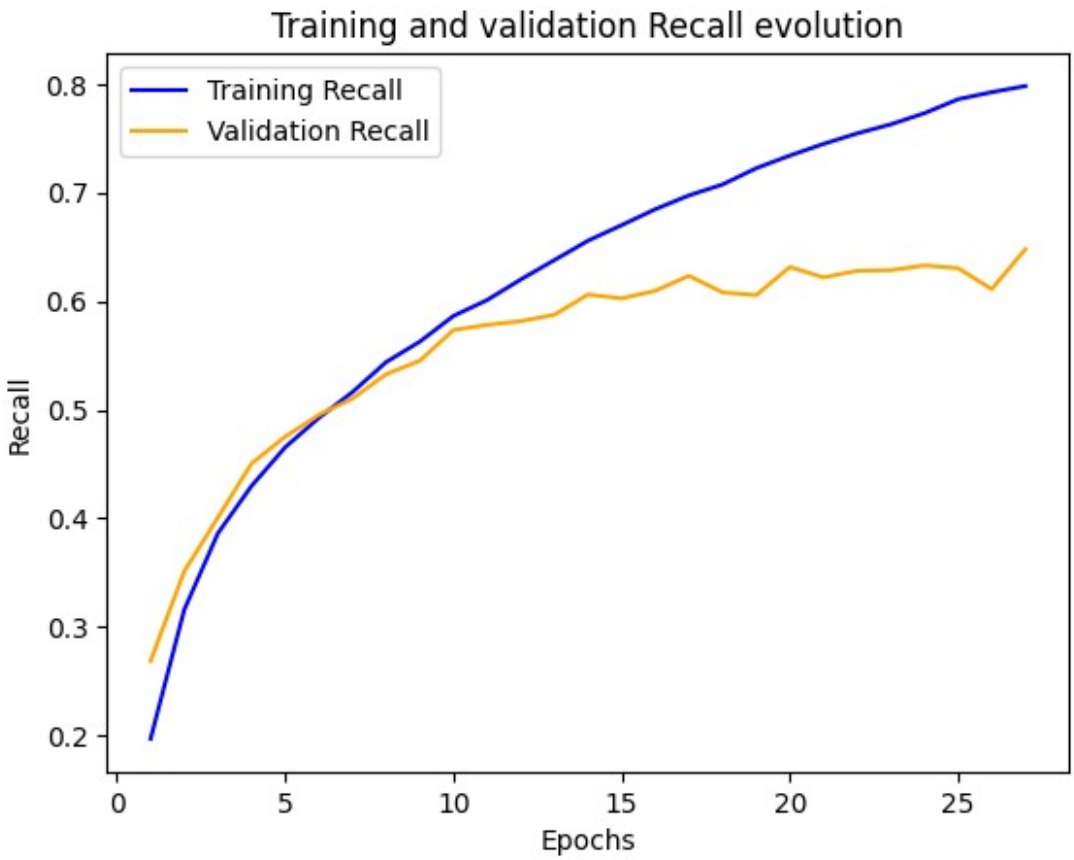
```
# Buscar as métricas
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
precision = history.history['precision']
val_precision = history.history['val_precision']
recall = history.history['recall']
val_recall = history.history['val_recall']

# Calcular o número de épocas que foram realizadas
epochs = range(1, len(acc) + 1)

# Gráfico da accuracy
plt.plot(epochs, acc, 'blue', label='Training Accuracy')
plt.plot(epochs, val_acc, 'orange', label='Validation Accuracy')
plt.title('Training and validation Accuracy evolution')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()

# Gráfico da loss
plt.plot(epochs, loss, 'blue', label='Training Loss')
plt.plot(epochs, val_loss, 'orange', label='Validation Loss')
```

```
plt.title('Training and validation Loss evolution')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.figure()

# Gráfico da precision
plt.plot(epochs, precision, 'blue', label='Training Precision')
plt.plot(epochs, val_precision, 'orange', label='Validation
Precision')
plt.title('Training and validation Precision evolution')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.figure()

# Gráfico do recall
plt.plot(epochs, recall, 'blue', label='Training Recall')
plt.plot(epochs, val_recall, 'orange', label='Validation Recall')
plt.title('Training and validation Recall evolution')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()

plt.show()
```


Training and validation Accuracy evolution
Training Accuracy
Validation Accuracy
Accuracy
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25
Epochs


Training and validation Loss evolution
Training Loss
Validation Loss
Loss
Epochs
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
5
10
15
20
25


Training and validation Precision evolution
Training Precision
Validation Precision
Precision
Epochs
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25



4.2 Desempenho no dataset de teste

De modo a compreender o real desempenho do modelo precisamos avaliar este utilizando o dataset de teste (que contém imagens que o este nunca viu anteriormente).

São feitas, e guardadas, previsões do modelo sobre o dataset de teste para, posteriormente, ser criado um classification report, que nos vai permitir analisar a taxa de acerto global e a precision, recall e f1-score para cada classe. Para além disso, é, também, construida uma matriz de confusão que, vai permitir ilustrar de uma outra maneira as previsões (vai ser possível ver, por exemplo, que quando a imagem pertencia à classe "dog", o modelo achou n vezes que a imagem pertencia à classe "cat").

Com isto, podemos compreender que:

- Os resultados são satisfatórios, apesar de existirem algumas preocupações com os valores obtidos nas métricas de recall, precision e f1-score das classes Cat, Bird e Dog
- O modelo identifica bem as seguintes classes:
  - Automobile
  - Frog
  - Ship
  - Truck
- O modelo mostra dificuldades em prever as seguintes classes:
  - Dog

  - Cat
- Existe uma clara dificuldade em distinguir as classes Dog e Cat

```
# Fazer previsões para o dataset de teste
predictions = model.predict(test_dataset)
predicted_classes = np.argmax(predictions, axis=1)

# Obter as classes verdadeiras de cada imagem no dataset de teste
true_classes = []
for images, labels in test_dataset:
    true_classes.extend(np.argmax(labels.numpy(), axis=1))
true_classes = np.array(true_classes)

# Criar o classification report
report = classification_report(true_classes, predicted_classes,
target_names=class_names)
print(report)

# Mostrar a matriz de confusão
cm = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
plt.show()
```

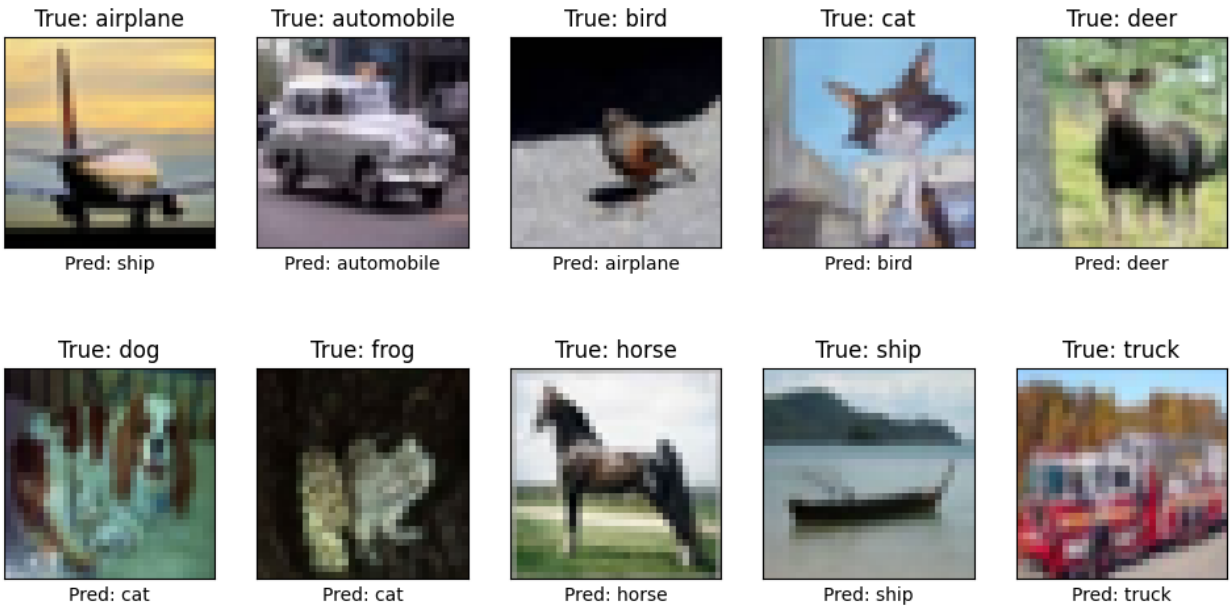
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
              precision    recall  f1-score   support

    airplane       0.72      0.75      0.74      1000
  automobile       0.87      0.75      0.81      1000
        bird       0.60      0.57      0.59      1000
         cat       0.60      0.48      0.53      1000
        deer       0.57      0.78      0.65      1000
         dog       0.67      0.56      0.61      1000
        frog       0.75      0.79      0.77      1000
       horse       0.77      0.71      0.74      1000
        ship       0.78      0.82      0.80      1000
       truck       0.74      0.84      0.79      1000

    accuracy                           0.70     10000
   macro avg       0.71      0.70      0.70     10000
weighted avg       0.71      0.70      0.70     10000
```

4.3 Visualização de previsões

Aqui fazemos a visualização de imagens tal como anteriormente, mas introduzimos a previsão do modelo para cada uma das imagens, sendo possível visualizar, também, a classe real de cada imagem.

```
displayed_classes = set()

plt.figure(figsize=(12, 6))  # Ajustar o tamanho das imagens

for data_batch, label_batch in test_dataset:
    for i in range(len(label_batch)):
        true_class_idx = np.argmax(label_batch[i])
        true_label = class_names[true_class_idx]

        if true_class_idx not in displayed_classes:
```

```
            displayed_classes.add(true_class_idx)

            plt.subplot(2, 5, len(displayed_classes))

            pred_label = model.predict(np.expand_dims(data_batch[i],
axis=0), verbose=0)
            pred_label = class_names[np.argmax(pred_label)]

            plt.title("True: " + true_label)
            plt.xlabel("Pred: " + pred_label)
            plt.imshow(data_batch[i].numpy().astype('uint8'))
            plt.xticks([])
            plt.yticks([])

        # Stop condition para no caso de já terem sido mostrada 10
imagens
        if len(displayed_classes) == 10:
            break
    if len(displayed_classes) == 10:
        break

plt.show()
```



Conclusões

O modelo tem um claro problema de overfitting. É expectável que, ao utilizar a técnica de Data Augmentation (aumentar a quantidade de imagens no dataset de treino) este problema possa vir a ser resolvido. De modo a combater este problema de uma forma mais completa poderia-se também aumentar a complexidade do modelo (e.g. utilizar mais camadas, mais filtros por

camada, ...) de modo a que, este seja capaz de aprender a identificar melhor as features que definem cada tipo de imagem.

Algumas das métricas obtidas no dataset de treino são preocupantes sendo que, ao aumentar a complexidade do modelo, tal como supramencionado, deverá corrigir este problema.

Bibliografia

https://www.markdownguide.org/basic-syntax/

https://www.tensorflow.org/

https://keras.io/api/applications/

https://keras.io/api/optimizers/

https://keras.io/api/data_loading/

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

https://nchlis.github.io/2017_08_10/page.html

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7