ImageClassification-AI: Modelo S

Versão B

Modelo de raiz com o intuito de obter os melhores resultados possíveis. Por consequência, este modelo vai almejar otimizar ao máximo a sua arquitetura.

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 # normalizar (feito mais à frente relo rescalling)
```

```
array([[[[0.4509804 , 0.58431375, 0.7490196 ],
         [0.4627451 , 0.59607846, 0.7647059 ],
         [0.4627451 , 0.6       , 0.7647059 ],
         ...,
         [0.39607844, 0.5372549 , 0.7529412 ],
         [0.39215687, 0.5294118 , 0.74509805],
```

```
         [0.3764706 , 0.5176471 , 0.73333335]],

        [[0.4862745 , 0.60784316, 0.7529412 ],
         [0.49411765, 0.6156863 , 0.7607843 ],
         [0.49803922, 0.61960787, 0.7647059 ],
         ...,
         [0.42352942, 0.5568628 , 0.75686276],
         [0.41568628, 0.54901963, 0.7490196 ],
         [0.4       , 0.53333336, 0.73333335]],

        [[0.5137255 , 0.61960787, 0.7411765 ],
         [0.5254902 , 0.6313726 , 0.7490196 ],
         [0.53333336, 0.6431373 , 0.7607843 ],
         ...,
         [0.4509804 , 0.5803922 , 0.7607843 ],
         [0.44705883, 0.57254905, 0.75686276],
         [0.43137255, 0.5568628 , 0.7411765 ]],

        ...,

        [[0.2627451 , 0.23137255, 0.1882353 ],
         [0.27058825, 0.23921569, 0.19607843],
         [0.2509804 , 0.21960784, 0.1764706 ],
         ...,
         [0.23921569, 0.23529412, 0.22745098],
         [0.24705882, 0.24313726, 0.23529412],
         [0.21960784, 0.21568628, 0.20784314]],

        [[0.27058825, 0.23921569, 0.19607843],
         [0.27450982, 0.24313726, 0.2       ],
         [0.23921569, 0.20784314, 0.16862746],
         ...,
         [0.22352941, 0.21960784, 0.21176471],
         [0.21568628, 0.21176471, 0.20392157],
         [0.22745098, 0.22352941, 0.21568628]],

        [[0.28235295, 0.2509804 , 0.20784314],
         [0.2627451 , 0.23137255, 0.1882353 ],
         [0.25490198, 0.22745098, 0.18431373],
         ...,
         [0.21176471, 0.20784314, 0.2       ],
         [0.2       , 0.19607843, 0.1882353 ],
         [0.21176471, 0.20784314, 0.2       ]]],


       [[[0.3372549 , 0.44313726, 0.3372549 ],
         [0.45882353, 0.6156863 , 0.45882353],
         [0.4509804 , 0.6313726 , 0.4392157 ],
         ...,
         [0.39607844, 0.5882353 , 0.42745098],
```

```
         [0.40392157, 0.58431375, 0.42745098],
         [0.4117647 , 0.5921569 , 0.43137255]],

        [[0.3254902 , 0.42352942, 0.3254902 ],
         [0.45490196, 0.6       , 0.45882353],
         [0.45490196, 0.62352943, 0.45490196],
         ...,
         [0.43529412, 0.654902  , 0.49803922],
         [0.4       , 0.6039216 , 0.44705883],
         [0.4       , 0.5882353 , 0.43137255]],

        [[0.32156864, 0.43137255, 0.3372549 ],
         [0.4509804 , 0.5882353 , 0.4745098 ],
         [0.3529412 , 0.49019608, 0.38039216],
         ...,
         [0.46666667, 0.69411767, 0.5647059 ],
         [0.44313726, 0.65882355, 0.5254902 ],
         [0.41568628, 0.6313726 , 0.49411765]],

        ...,

        [[0.5058824 , 0.7294118 , 0.47843137],
         [0.5176471 , 0.73333335, 0.49019608],
         [0.5372549 , 0.74509805, 0.49411765],
         ...,
         [0.39215687, 0.5803922 , 0.4862745 ],
         [0.4       , 0.5921569 , 0.46666667],
         [0.39215687, 0.58431375, 0.42745098]],

        [[0.38039216, 0.56078434, 0.35686275],
         [0.39215687, 0.5686275 , 0.36862746],
         [0.4117647 , 0.59607846, 0.38039216],
         ...,
         [0.36862746, 0.49411765, 0.4       ],
         [0.45882353, 0.59607846, 0.45882353],
         [0.40784314, 0.5764706 , 0.39215687]],

        [[0.3529412 , 0.5058824 , 0.34901962],
         [0.36078432, 0.5058824 , 0.35686275],
         [0.3647059 , 0.5176471 , 0.34901962],
         ...,
         [0.4627451 , 0.57254905, 0.45882353],
         [0.47843137, 0.6039216 , 0.46666667],
         [0.4117647 , 0.5764706 , 0.39607844]]],


       [[[0.6039216 , 0.5686275 , 0.7019608 ],
         [0.5647059 , 0.5411765 , 0.6627451 ],
         [0.4862745 , 0.4745098 , 0.5647059 ],
         ...,
```

```
         [0.02352941, 0.01960784, 0.01960784],
         [0.03137255, 0.02745098, 0.02745098],
         [0.03921569, 0.03529412, 0.03529412]],

        [[0.41960785, 0.40392157, 0.50980395],
         [0.37254903, 0.36862746, 0.45490196],
         [0.36078432, 0.35686275, 0.40784314],
         ...,
         [0.05098039, 0.03921569, 0.02745098],
         [0.06666667, 0.05490196, 0.04705882],
         [0.0627451 , 0.05098039, 0.03921569]],

        [[0.3529412 , 0.35686275, 0.4117647 ],
         [0.29411766, 0.3137255 , 0.34117648],
         [0.27450982, 0.28235295, 0.27058825],
         ...,
         [0.0627451 , 0.04313726, 0.02745098],
         [0.07843138, 0.05882353, 0.04313726],
         [0.0627451 , 0.04705882, 0.03529412]],

        ...,

        [[0.31764707, 0.32156864, 0.2901961 ],
         [0.37254903, 0.37254903, 0.3647059 ],
         [0.3882353 , 0.3882353 , 0.39607844],
         ...,
         [0.5921569 , 0.58431375, 0.5803922 ],
         [0.5882353 , 0.57254905, 0.5764706 ],
         [0.56078434, 0.54509807, 0.54901963]],

        [[0.24313726, 0.24705882, 0.21176471],
         [0.34117648, 0.34117648, 0.32156864],
         [0.3647059 , 0.3647059 , 0.35686275],
         ...,
         [0.3529412 , 0.34901962, 0.36078432],
         [0.34117648, 0.34117648, 0.34509805],
         [0.32156864, 0.31764707, 0.3254902 ]],

        [[0.29411766, 0.29803923, 0.28235295],
         [0.3137255 , 0.3137255 , 0.3137255 ],
         [0.35686275, 0.35686275, 0.35686275],
         ...,
         [0.37254903, 0.3764706 , 0.36078432],
         [0.39215687, 0.38039216, 0.3647059 ],
         [0.41568628, 0.40392157, 0.3882353 ]]],


       ...,
```

```
       [[[0.2509804 , 0.24313726, 0.22352941],
         [0.26666668, 0.25882354, 0.23921569],
         [0.2784314 , 0.27058825, 0.2509804 ],
         ...,
         [0.6       , 0.5803922 , 0.5294118 ],
         [0.627451  , 0.54901963, 0.4117647 ],
         [0.68235296, 0.5882353 , 0.42352942]],

        [[0.6313726 , 0.6117647 , 0.5921569 ],
         [0.5803922 , 0.56078434, 0.5411765 ],
         [0.53333336, 0.5137255 , 0.49411765],
         ...,
         [0.7137255 , 0.7058824 , 0.6666667 ],
         [0.6039216 , 0.52156866, 0.38039216],
         [0.6745098 , 0.5803922 , 0.41568628]],

        [[0.8235294 , 0.8156863 , 0.8117647 ],
         [0.8156863 , 0.80784315, 0.8039216 ],
         [0.8117647 , 0.8039216 , 0.8       ],
         ...,
         [0.84313726, 0.83137256, 0.8       ],
         [0.627451  , 0.5411765 , 0.41568628],
         [0.67058825, 0.57254905, 0.4117647 ]],

        ...,

        [[0.29803923, 0.28235295, 0.24705882],
         [0.13333334, 0.12156863, 0.09803922],
         [0.1254902 , 0.11764706, 0.09019608],
         ...,
         [0.16470589, 0.12941177, 0.07843138],
         [0.38431373, 0.28235295, 0.13725491],
         [0.41568628, 0.28235295, 0.10980392]],

        [[0.2       , 0.17254902, 0.14901961],
         [0.16470589, 0.14901961, 0.11764706],
         [0.17254902, 0.16470589, 0.11764706],
         ...,
         [0.19607843, 0.1764706 , 0.11764706],
         [0.30980393, 0.25490198, 0.14117648],
         [0.38431373, 0.27058825, 0.11764706]],

        [[0.4117647 , 0.4       , 0.3764706 ],
         [0.3764706 , 0.3647059 , 0.3254902 ],
         [0.27058825, 0.2627451 , 0.21176471],
         ...,
         [0.1764706 , 0.1764706 , 0.14117648],
         [0.19607843, 0.1882353 , 0.13333334],
         [0.27058825, 0.21960784, 0.12156863]]],
```

```
       [[[0.37254903, 0.34901962, 0.25490198],
         [0.36862746, 0.34509805, 0.25490198],
         [0.3764706 , 0.3529412 , 0.25882354],
         ...,
         [0.43529412, 0.42745098, 0.42745098],
         [0.4392157 , 0.4392157 , 0.4509804 ],
         [0.40784314, 0.42352942, 0.4392157 ]],

        [[0.38039216, 0.35686275, 0.25882354],
         [0.3764706 , 0.3529412 , 0.25882354],
         [0.38039216, 0.35686275, 0.2627451 ],
         ...,
         [0.30980393, 0.30588236, 0.29803923],
         [0.2901961 , 0.2901961 , 0.3019608 ],
         [0.25490198, 0.27450982, 0.28627452]],

        [[0.3764706 , 0.3529412 , 0.25882354],
         [0.37254903, 0.34901962, 0.25490198],
         [0.3764706 , 0.3529412 , 0.25882354],
         ...,
         [0.28235295, 0.28235295, 0.26666668],
         [0.2627451 , 0.26666668, 0.27450982],
         [0.22745098, 0.24313726, 0.25490198]],

        ...,

        [[0.8352941 , 0.8509804 , 0.84705883],
         [0.83137256, 0.84705883, 0.84313726],
         [0.85490197, 0.87058824, 0.8666667 ],
         ...,
         [0.3137255 , 0.30980393, 0.28627452],
         [0.21960784, 0.21568628, 0.20392157],
         [0.16862746, 0.16862746, 0.16862746]],

        [[0.83137256, 0.84705883, 0.84313726],
         [0.84313726, 0.85882354, 0.85490197],
         [0.84705883, 0.8627451 , 0.85882354],
         ...,
         [0.2901961 , 0.2901961 , 0.2627451 ],
         [0.21960784, 0.21568628, 0.20392157],
         [0.1764706 , 0.1764706 , 0.18039216]],

        [[0.73333335, 0.7490196 , 0.74509805],
         [0.75686276, 0.77254903, 0.76862746],
         [0.78431374, 0.8       , 0.79607844],
         ...,
         [0.25490198, 0.2509804 , 0.22352941],
         [0.21960784, 0.21568628, 0.20392157],
         [0.18431373, 0.18431373, 0.18431373]]],
```

```
       [[[0.65882355, 0.10588235, 0.04313726],
         [0.654902  , 0.10980392, 0.06666667],
         [0.654902  , 0.10980392, 0.05490196],
         ...,
         [0.60784316, 0.07843138, 0.02352941],
         [0.60784316, 0.07843138, 0.02745098],
         [0.6156863 , 0.07450981, 0.01960784]],

        [[0.6745098 , 0.10588235, 0.05882353],
         [0.67058825, 0.10588235, 0.06666667],
         [0.67058825, 0.10980392, 0.05098039],
         ...,
         [0.6156863 , 0.07843138, 0.02352941],
         [0.6117647 , 0.07843138, 0.02352941],
         [0.6156863 , 0.07058824, 0.01568628]],

        [[0.67058825, 0.10196079, 0.0627451 ],
         [0.6745098 , 0.10588235, 0.0627451 ],
         [0.6627451 , 0.10980392, 0.05098039],
         ...,
         [0.6156863 , 0.08235294, 0.02745098],
         [0.60784316, 0.07450981, 0.01960784],
         [0.6117647 , 0.07058824, 0.01568628]],

        ...,

        [[0.85882354, 0.8117647 , 0.827451  ],
         [0.8666667 , 0.8235294 , 0.827451  ],
         [0.8862745 , 0.80784315, 0.8235294 ],
         ...,
         [0.8392157 , 0.29803923, 0.3647059 ],
         [0.8235294 , 0.2784314 , 0.3372549 ],
         [0.83137256, 0.2784314 , 0.32156864]],

        [[0.78431374, 0.53333336, 0.54901963],
         [0.7647059 , 0.5411765 , 0.54509807],
         [0.8117647 , 0.5176471 , 0.5647059 ],
         ...,
         [0.8392157 , 0.34901962, 0.43137255],
         [0.85490197, 0.3372549 , 0.42352942],
         [0.8666667 , 0.32156864, 0.39607844]],

        [[0.7647059 , 0.3019608 , 0.3137255 ],
         [0.77254903, 0.3019608 , 0.3372549 ],
         [0.8156863 , 0.3372549 , 0.4117647 ],
         ...,
         [0.8627451 , 0.37254903, 0.48235294],
```

```
        [0.8666667 , 0.3647059 , 0.47843137],
        [0.8627451 , 0.34509805, 0.44313726]]]], 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

A arquitetura deste modelo foi inspirada na arquitetura do modelo VGG16. Com isto, temos:

- Como supramencionado, a normalização dos valores de cada pixel da imagem
- Três blocos de layers convolucionais:

  - Cada um com duas camadas convolucionais:
  - A quantidade de filtros em cada camada convolucional vai aumentado progressivamente de 32 filtros até 128 e mantem-se constante dentro de cada bloco convolucional, isto é, dentro do mesmo bloco os filtros é utilizada a mesma quantidade de filtros para as ambas as camadas
  - É utilizada a função de ativação ReLu
  - No final de cada bloco convolucional 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)
  - Ainda sobre o final de cada bloco convolucional, é utilizado o Dropout() com o valor de 0.2, isto é, ou seja, no final de cada bloco convolucional são excluidos 20% (x * 100 %, sendo x o valor do paramêtro utilizado no Dropout) dos neurónios presentes naquele momento
  - 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
  - É utilizada uma outra camada densa, com 10 filtros (que equivalem ao número de classes presentes no 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 (2 layers)
x = layers.Conv2D(filters=32, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters=32, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)
```

```
# 2nd Convolutional Block (2 layers)
x = layers.Conv2D(filters=64, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters=64, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)

# 3rd Convolutional Block (2 layers)
x = layers.Conv2D(filters=128, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.Conv2D(filters=128, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.2)(x)

# Classification Block
x = layers.Flatten()(x)
x = layers.Dense(128)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(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, 32, 32, 32)      |
896 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| batch_normalization               | (None, 32, 32, 32)      |
128 |
| (BatchNormalization)              |                         |
|
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| activation (Activation)           | (None, 32, 32, 32)      |
0 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| conv2d_1 (Conv2D)                 | (None, 32, 32, 32)      |
9,248 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| batch_normalization_1             | (None, 32, 32, 32)      |
128 |
| (BatchNormalization)              |                         |
|
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| activation_1 (Activation)         | (None, 32, 32, 32)      |
0 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| max_pooling2d (MaxPooling2D)      | (None, 16, 16, 32)      |
0 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| dropout (Dropout)                 | (None, 16, 16, 32)      |
0 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| conv2d_2 (Conv2D)                 | (None, 16, 16, 64)      |
18,496 |
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| batch_normalization_2             | (None, 16, 16, 64)      |
256 |
| (BatchNormalization)              |                         |
|
├──────────────────────────────────┼──────────────────────────┼──────────
──────┤
| activation_2 (Activation)         | (None, 16, 16, 64)      |
0 |
```

```
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| conv2d_3 (Conv2D)                 | (None, 16, 16, 64)          |
36,928 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_3             | (None, 16, 16, 64)          |
256 |
| (BatchNormalization)              |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_3 (Activation)         | (None, 16, 16, 64)          |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| max_pooling2d_1 (MaxPooling2D)    | (None, 8, 8, 64)            |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| dropout_1 (Dropout)               | (None, 8, 8, 64)            |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| conv2d_4 (Conv2D)                 | (None, 8, 8, 128)           |
73,856 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_4             | (None, 8, 8, 128)           |
512 |
| (BatchNormalization)              |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_4 (Activation)         | (None, 8, 8, 128)           |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| conv2d_5 (Conv2D)                 | (None, 8, 8, 128)           |
147,584 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_5             | (None, 8, 8, 128)           |
512 |
| (BatchNormalization)              |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_5 (Activation)         | (None, 8, 8, 128)           |
```

```
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 4, 4, 128)       │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dropout_2 (Dropout)             │ (None, 4, 4, 128)       │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ flatten (Flatten)               │ (None, 2048)            │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense (Dense)                   │ (None, 128)             │
262,272 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ batch_normalization_6           │ (None, 128)             │
512 │
│ (BatchNormalization)            │                         │
│
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ activation_6 (Activation)       │ (None, 128)             │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dropout_3 (Dropout)             │ (None, 128)             │
0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                 │ (None, 10)              │
1,290 │
└──────────────────────────────────────┴─────────────────────────────┴──────────
──────┘

 Total params: 552,874 (2.11 MB)

 Trainable params: 551,722 (2.10 MB)

 Non-trainable params: 1,152 (4.50 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.

Neste modelo foi utilizado como otimizador o Adam, com o principal objetivo de explorar mais otimizadores. Não é definido um learning rate a ser utilizado por este otimizador, sendo utilizado o rate por omissão, já que este já possui, de base, técnicas de otimização do learning rate.

```
model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizers.Adam(),
    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_B.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 57ms/step - accuracy: 0.3748 - loss:
1.8981 - precision: 0.5323 - recall: 0.1796
Epoch 1: val_loss improved from inf to 2.18712, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 82s 61ms/step - accuracy: 0.3749 -
loss: 1.8976 - precision: 0.5325 - recall: 0.1797 - val_accuracy:
0.3518 - val_loss: 2.1871 - val_precision: 0.4578 - val_recall: 0.2430
Epoch 2/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.6165 - loss:
1.1683 - precision: 0.7774 - recall: 0.4405
Epoch 2: val_loss improved from 2.18712 to 1.48422, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 81s 65ms/step - accuracy: 0.6165 -
loss: 1.1682 - precision: 0.7775 - recall: 0.4405 - val_accuracy:
0.5268 - val_loss: 1.4842 - val_precision: 0.6353 - val_recall: 0.4321
Epoch 3/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.6811 - loss:
0.9808 - precision: 0.8036 - recall: 0.5500
Epoch 3: val_loss improved from 1.48422 to 1.31212, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 81s 65ms/step - accuracy: 0.6811 -
loss: 0.9807 - precision: 0.8036 - recall: 0.5500 - val_accuracy:
0.5762 - val_loss: 1.3121 - val_precision: 0.6828 - val_recall: 0.4895
Epoch 4/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.7188 - loss:
0.8774 - precision: 0.8259 - recall: 0.6094
Epoch 4: val_loss improved from 1.31212 to 0.86177, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.7188 -
loss: 0.8773 - precision: 0.8259 - recall: 0.6094 - val_accuracy:
0.7220 - val_loss: 0.8618 - val_precision: 0.8097 - val_recall: 0.6477
Epoch 5/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - accuracy: 0.7486 - loss:
0.7878 - precision: 0.8429 - recall: 0.6595
Epoch 5: val_loss did not improve from 0.86177
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 96s 76ms/step - accuracy: 0.7486 -
loss: 0.7878 - precision: 0.8429 - recall: 0.6595 - val_accuracy:
0.6837 - val_loss: 0.9861 - val_precision: 0.7680 - val_recall: 0.6143
```

```
Epoch 6/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.7700 - loss:
0.7364 - precision: 0.8492 - recall: 0.6895
Epoch 6: val_loss improved from 0.86177 to 0.83671, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 82s 65ms/step - accuracy: 0.7701 -
loss: 0.7364 - precision: 0.8492 - recall: 0.6895 - val_accuracy:
0.7294 - val_loss: 0.8367 - val_precision: 0.7999 - val_recall: 0.6705
Epoch 7/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.7905 - loss:
0.6660 - precision: 0.8607 - recall: 0.7172
Epoch 7: val_loss improved from 0.83671 to 0.70870, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 85s 68ms/step - accuracy: 0.7905 -
loss: 0.6660 - precision: 0.8607 - recall: 0.7172 - val_accuracy:
0.7729 - val_loss: 0.7087 - val_precision: 0.8284 - val_recall: 0.7220
Epoch 8/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.8071 - loss:
0.6243 - precision: 0.8733 - recall: 0.7423
Epoch 8: val_loss improved from 0.70870 to 0.61881, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 86s 68ms/step - accuracy: 0.8071 -
loss: 0.6243 - precision: 0.8733 - recall: 0.7423 - val_accuracy:
0.8043 - val_loss: 0.6188 - val_precision: 0.8576 - val_recall: 0.7629
Epoch 9/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8136 - loss:
0.5960 - precision: 0.8763 - recall: 0.7541
Epoch 9: val_loss did not improve from 0.61881
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 78s 63ms/step - accuracy: 0.8136 -
loss: 0.5960 - precision: 0.8763 - recall: 0.7541 - val_accuracy:
0.7985 - val_loss: 0.6310 - val_precision: 0.8579 - val_recall: 0.7513
Epoch 10/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.8251 - loss:
0.5610 - precision: 0.8845 - recall: 0.7734
Epoch 10: val_loss did not improve from 0.61881
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 77s 62ms/step - accuracy: 0.8251 -
loss: 0.5610 - precision: 0.8845 - recall: 0.7734 - val_accuracy:
0.7870 - val_loss: 0.6835 - val_precision: 0.8388 - val_recall: 0.7437
Epoch 11/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.8372 - loss:
0.5254 - precision: 0.8918 - recall: 0.7898
Epoch 11: val_loss improved from 0.61881 to 0.61135, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 87s 70ms/step - accuracy: 0.8372 -
loss: 0.5254 - precision: 0.8918 - recall: 0.7898 - val_accuracy:
0.8101 - val_loss: 0.6114 - val_precision: 0.8622 - val_recall: 0.7653
Epoch 12/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.8475 - loss:
0.4942 - precision: 0.8942 - recall: 0.8022
```

```
Epoch 12: val_loss did not improve from 0.61135
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.8476 -
loss: 0.4942 - precision: 0.8942 - recall: 0.8023 - val_accuracy:
0.7896 - val_loss: 0.6930 - val_precision: 0.8321 - val_recall: 0.7512
Epoch 13/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.8577 - loss:
0.4697 - precision: 0.9028 - recall: 0.8151
Epoch 13: val_loss did not improve from 0.61135
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 93s 74ms/step - accuracy: 0.8577 -
loss: 0.4697 - precision: 0.9028 - recall: 0.8151 - val_accuracy:
0.7983 - val_loss: 0.6591 - val_precision: 0.8436 - val_recall: 0.7648
Epoch 14/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8626 - loss:
0.4474 - precision: 0.9034 - recall: 0.8256
Epoch 14: val_loss improved from 0.61135 to 0.55261, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 78s 62ms/step - accuracy: 0.8626 -
loss: 0.4474 - precision: 0.9034 - recall: 0.8256 - val_accuracy:
0.8293 - val_loss: 0.5526 - val_precision: 0.8747 - val_recall: 0.7929
Epoch 15/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8699 - loss:
0.4301 - precision: 0.9070 - recall: 0.8305
Epoch 15: val_loss did not improve from 0.55261
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 73s 58ms/step - accuracy: 0.8699 -
loss: 0.4301 - precision: 0.9070 - recall: 0.8305 - val_accuracy:
0.8195 - val_loss: 0.5851 - val_precision: 0.8620 - val_recall: 0.7869
Epoch 16/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.8791 - loss:
0.4029 - precision: 0.9145 - recall: 0.8452
Epoch 16: val_loss did not improve from 0.55261
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 78s 62ms/step - accuracy: 0.8791 -
loss: 0.4029 - precision: 0.9145 - recall: 0.8452 - val_accuracy:
0.8316 - val_loss: 0.5567 - val_precision: 0.8685 - val_recall: 0.8036
Epoch 17/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8799 - loss:
0.3992 - precision: 0.9138 - recall: 0.8472
Epoch 17: val_loss did not improve from 0.55261
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 85s 68ms/step - accuracy: 0.8799 -
loss: 0.3992 - precision: 0.9138 - recall: 0.8472 - val_accuracy:
0.8244 - val_loss: 0.5689 - val_precision: 0.8620 - val_recall: 0.8008
Epoch 18/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8834 - loss:
0.3830 - precision: 0.9162 - recall: 0.8540
Epoch 18: val_loss did not improve from 0.55261
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 86s 69ms/step - accuracy: 0.8834 -
loss: 0.3829 - precision: 0.9162 - recall: 0.8540 - val_accuracy:
0.8350 - val_loss: 0.5619 - val_precision: 0.8629 - val_recall: 0.8142
Epoch 19/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.8906 - loss:
```

```
0.3575 - precision: 0.9228 - recall: 0.8647
Epoch 19: val_loss improved from 0.55261 to 0.49980, saving model to
models/IC_S_B.keras
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.8906 -
loss: 0.3575 - precision: 0.9228 - recall: 0.8647 - val_accuracy:
0.8461 - val_loss: 0.4998 - val_precision: 0.8775 - val_recall: 0.8246
Epoch 20/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.8936 - loss:
0.3495 - precision: 0.9238 - recall: 0.8673
Epoch 20: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 82s 65ms/step - accuracy: 0.8936 -
loss: 0.3495 - precision: 0.9238 - recall: 0.8673 - val_accuracy:
0.8223 - val_loss: 0.5889 - val_precision: 0.8519 - val_recall: 0.7986
Epoch 21/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.8965 - loss:
0.3438 - precision: 0.9227 - recall: 0.8709
Epoch 21: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 90s 72ms/step - accuracy: 0.8965 -
loss: 0.3438 - precision: 0.9227 - recall: 0.8709 - val_accuracy:
0.8395 - val_loss: 0.5530 - val_precision: 0.8656 - val_recall: 0.8201
Epoch 22/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9023 - loss:
0.3286 - precision: 0.9263 - recall: 0.8796
Epoch 22: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 86s 69ms/step - accuracy: 0.9023 -
loss: 0.3286 - precision: 0.9263 - recall: 0.8796 - val_accuracy:
0.8402 - val_loss: 0.5411 - val_precision: 0.8668 - val_recall: 0.8244
Epoch 23/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9044 - loss:
0.3163 - precision: 0.9282 - recall: 0.8810
Epoch 23: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 84s 67ms/step - accuracy: 0.9044 -
loss: 0.3163 - precision: 0.9282 - recall: 0.8810 - val_accuracy:
0.8502 - val_loss: 0.5182 - val_precision: 0.8723 - val_recall: 0.8319
Epoch 24/100
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 0.9071 - loss:
0.3142 - precision: 0.9318 - recall: 0.8839
Epoch 24: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 88s 71ms/step - accuracy: 0.9071 -
loss: 0.3142 - precision: 0.9318 - recall: 0.8839 - val_accuracy:
0.8334 - val_loss: 0.5798 - val_precision: 0.8586 - val_recall: 0.8140
Epoch 25/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 0.9108 - loss:
0.3026 - precision: 0.9327 - recall: 0.8903
Epoch 25: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 89s 71ms/step - accuracy: 0.9108 -
loss: 0.3026 - precision: 0.9327 - recall: 0.8903 - val_accuracy:
0.8396 - val_loss: 0.5583 - val_precision: 0.8629 - val_recall: 0.8214
Epoch 26/100
```

```
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9102 - loss:
0.3002 - precision: 0.9320 - recall: 0.8895
Epoch 26: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 86s 69ms/step - accuracy: 0.9102 -
loss: 0.3002 - precision: 0.9320 - recall: 0.8895 - val_accuracy:
0.8422 - val_loss: 0.5358 - val_precision: 0.8666 - val_recall: 0.8252
Epoch 27/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9134 - loss:
0.2853 - precision: 0.9344 - recall: 0.8957
Epoch 27: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.9134 -
loss: 0.2853 - precision: 0.9344 - recall: 0.8957 - val_accuracy:
0.8477 - val_loss: 0.5529 - val_precision: 0.8694 - val_recall: 0.8326
Epoch 28/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 0.9180 - loss:
0.2801 - precision: 0.9373 - recall: 0.8988
Epoch 28: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 89s 71ms/step - accuracy: 0.9180 -
loss: 0.2801 - precision: 0.9373 - recall: 0.8988 - val_accuracy:
0.8484 - val_loss: 0.5257 - val_precision: 0.8728 - val_recall: 0.8327
Epoch 29/100
1251/1252 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.9188 - loss:
0.2728 - precision: 0.9376 - recall: 0.9000
Epoch 29: val_loss did not improve from 0.49980
1252/1252 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.9188 -
loss: 0.2728 - precision: 0.9376 - recall: 0.9000 - val_accuracy:
0.8531 - val_loss: 0.5207 - val_precision: 0.8756 - val_recall: 0.8358
```

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.

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

# Avaliar o modelo
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 ━━━━━━━━━━━━━━━━━━━━ 7s 22ms/step - accuracy: 0.8549 - loss:
0.4925 - precision: 0.8801 - recall: 0.8323
Test Accuracy: 0.8458999991416931
```

```
Test Loss: 0.5175831913948059
Test Precision: 0.8727716207504272
Test Recall: 0.8224999904632568
```

4. Análise dos resultados

4.1 Evolução das métricas durante o processo de treino

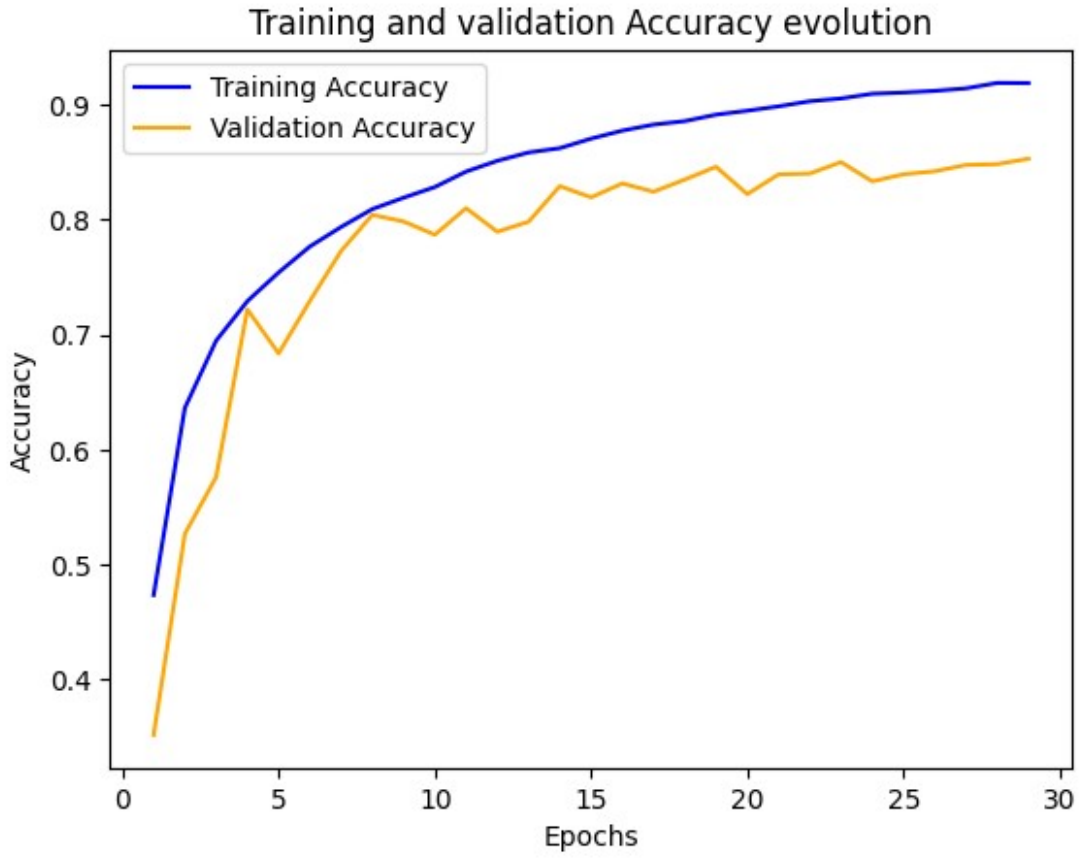
São utilizados gráficos para melhor compreender de que maneira as métricas, nomeadamente a accuracy, loss, precision e recall, foram evoluindo ao longo do processo de treino.

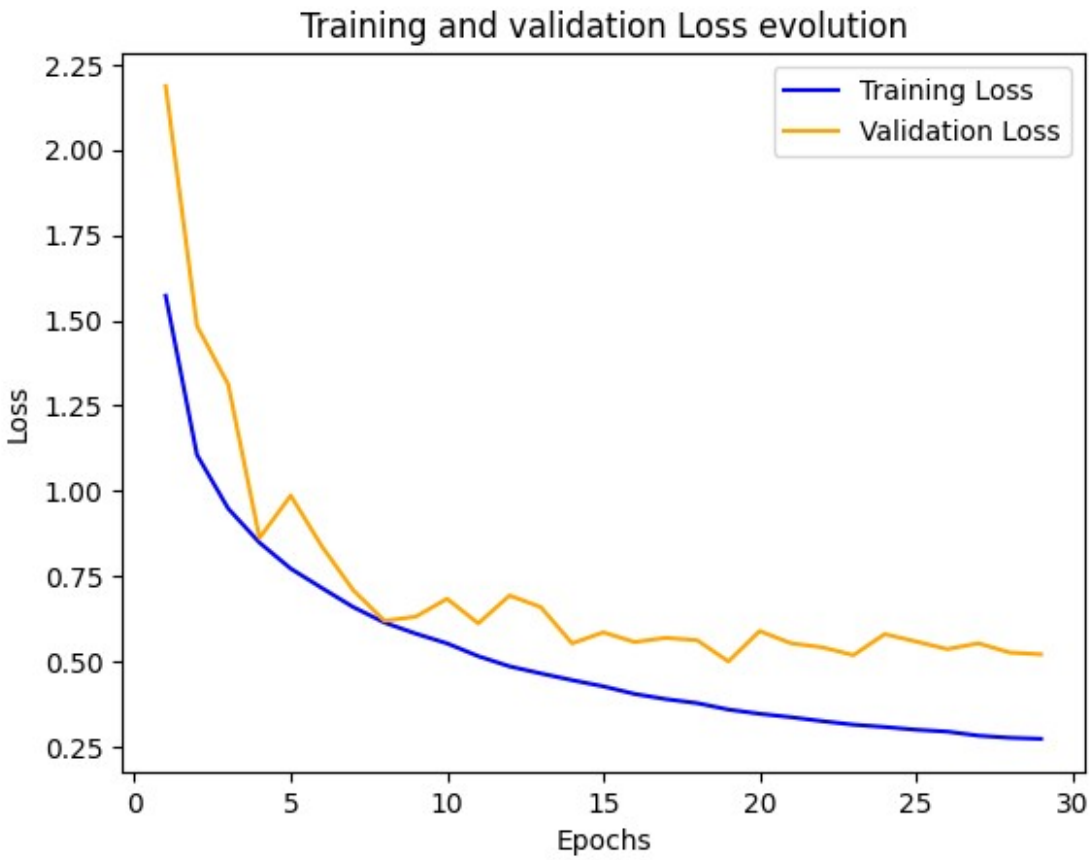
É possível visualizar que:

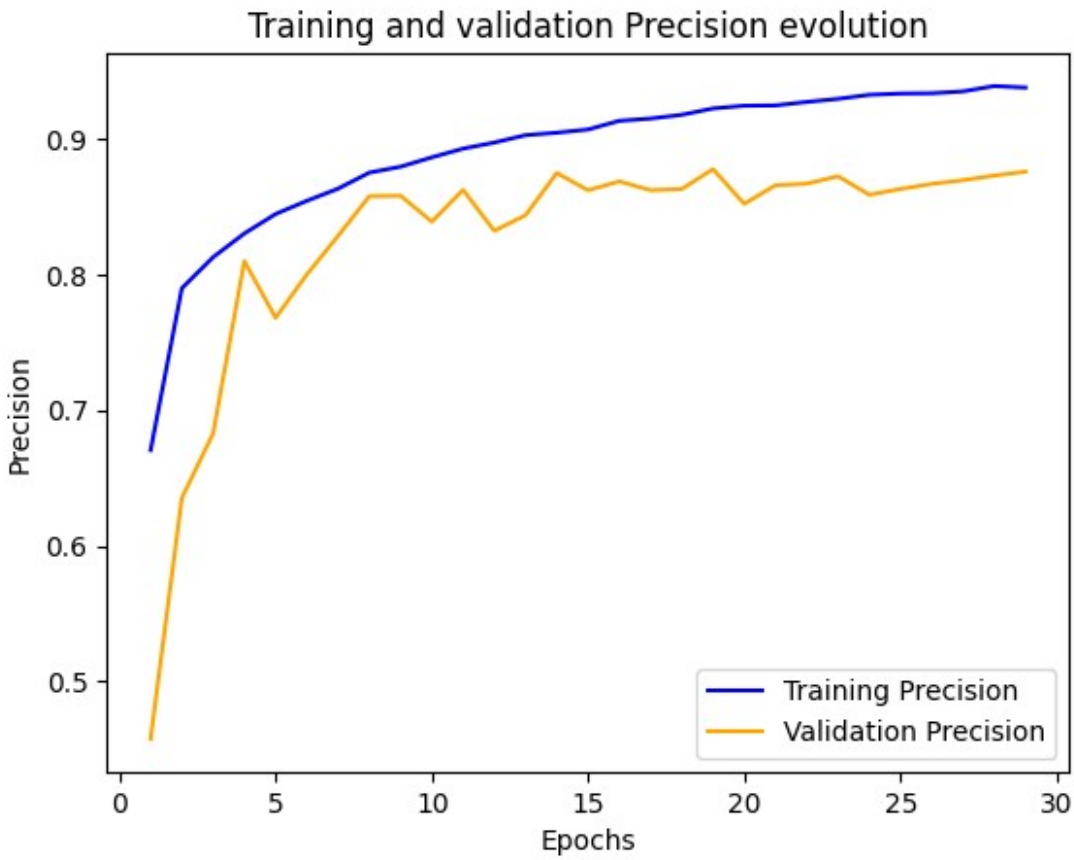
- O modelo começa a entrar em overfitting por volta da época 20

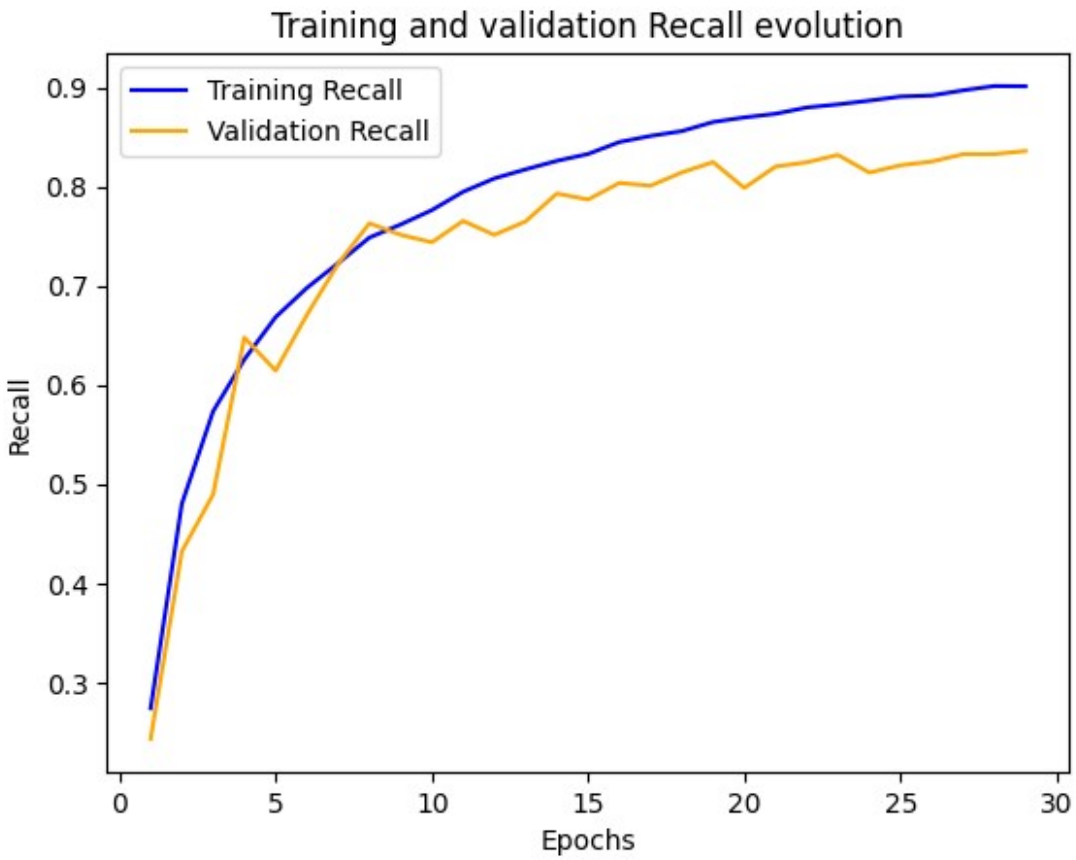
```
# Buscar as métricas
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
precision = history.history['precision']
val_precision = history.history['val_precision']
recall = history.history['recall']
val_recall = 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 Loss evolution
Training Loss
Validation Loss
Loss
Epochs
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
5
10
15
20
25
30


Training and validation Precision evolution
Precision
Epochs
0.5
0.6
0.7
0.8
0.9
0
5
10
15
20
25
30
Training Precision
Validation Precision



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).

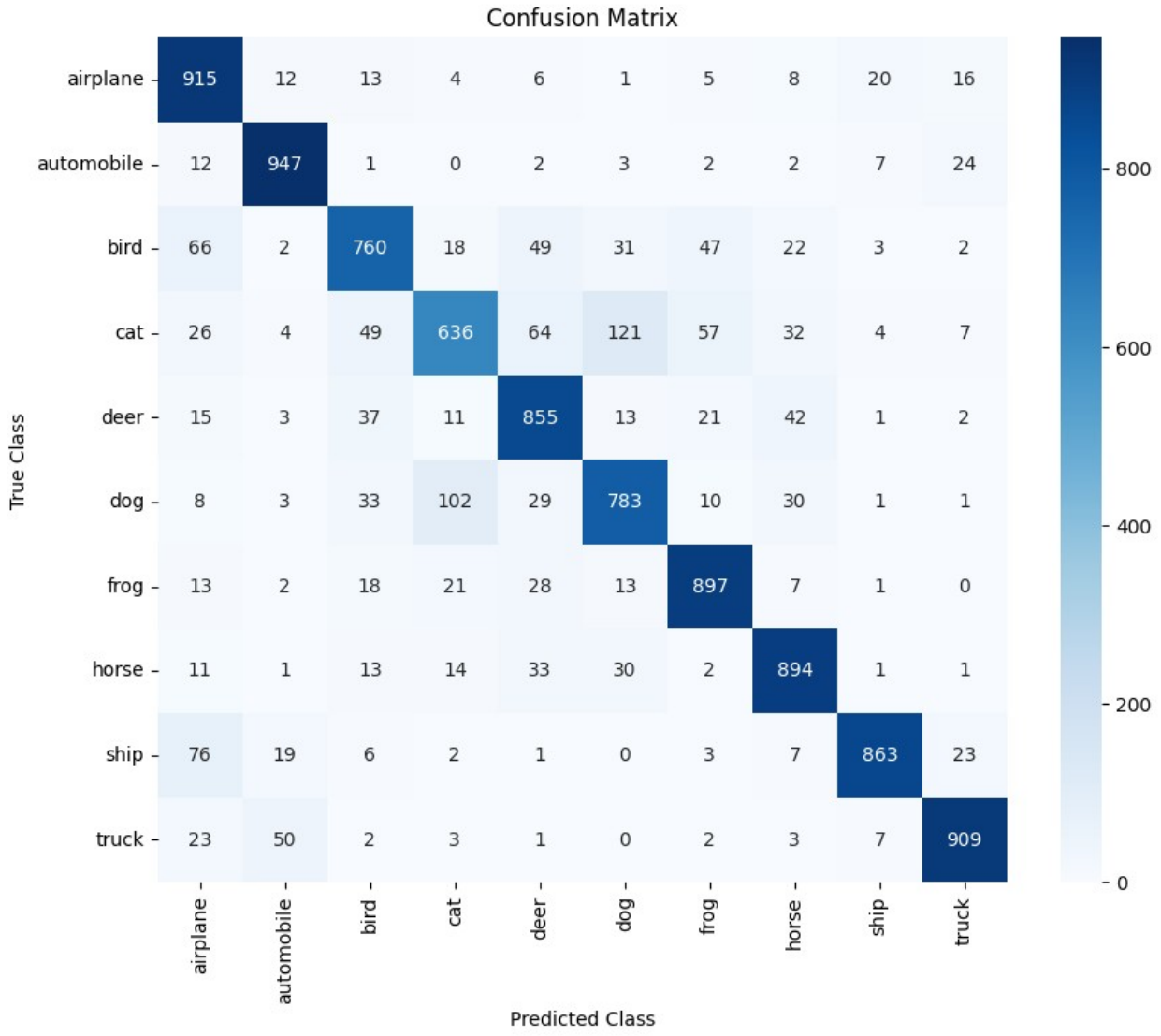
São feitas, e guardadas, previsões do modelo sobre o dataset de teste para, posteriormente, ser criado um classification report, que nos vai permitir analisar a taxa de acerto global e a precision, recall e f1-score para cada classe. Para além disso, é, também, construida uma matriz de confusão que, 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:

- O modelo obtem resultados bastante satisfatórios
- É possível perceber que existe alguma dificuldade em classificar certas classes, especialmente quando comparamos os resultados obtidos nestas com os obtidos nas outras. Estas classes são:
  - Bird
  - Cat
  - Dog

```python
# 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 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step
              precision    recall  f1-score   support

    airplane       0.79      0.92      0.85      1000
  automobile       0.91      0.95      0.93      1000
        bird       0.82      0.76      0.79      1000
         cat       0.78      0.64      0.70      1000
        deer       0.80      0.85      0.83      1000
         dog       0.79      0.78      0.78      1000
        frog       0.86      0.90      0.88      1000
       horse       0.85      0.89      0.87      1000
        ship       0.95      0.86      0.90      1000
       truck       0.92      0.91      0.92      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.84     10000
weighted avg       0.85      0.85      0.84     10000
```

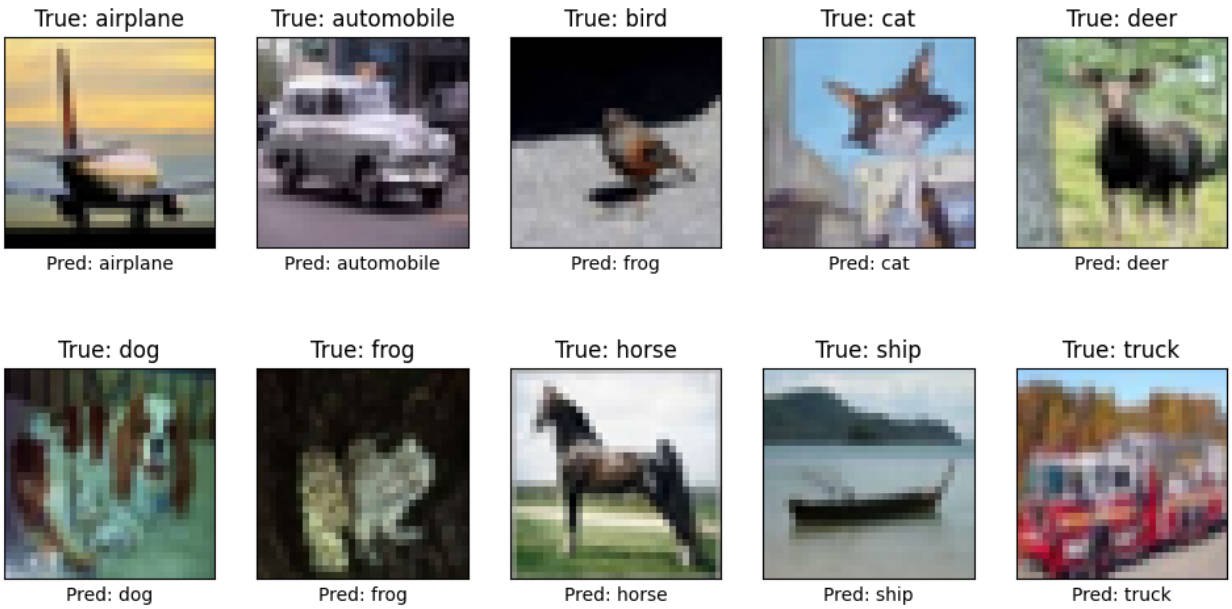


4.3 Visualização de previsões

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 pequeno problema de overfitting já que, a partir da época 20, não há nenhuma melhoria. É expectável que ao treinar o modelo utilizando a técnica de Data Augmentation este problema acabe por ser mitigado.

Também relacionado com o overfitting está o espaço para melhoria que existe na classificação das classes Bird, Cat e Dog. Apesar de ser considerar que os resultados obtidos para essas

classes são satisfatórios existe, um claro espaço para melhoria neste aspeto do modelo. Uma das maneiras que em que isto pode ser realizado é com a utilização de mais camadas convolucionais ou de mais filtros na arquitetura do modelo, com o objetivo de aumentar a capacidade de reconhecimento de features especificas das classes mencionadas por parte do modelo.

É importante realçar que, o modelo também foi treinado utilizando um scheduler para o learning rate e o Optuna, um algoritmo de otimização dos hyperparameters. Os resultados que foram obtidos nestes treinos não foram satisfatórios e, foi tomada a decisão de remover estes dois do modelo final. Em ambos os casos o modelo não melhorava significativamente, sendo que, no caso do scheduler, verificámos que o modelo perdia demasiada capacidade de convergência e, no caso do Optuna, o modelo não foi capaz de obter resultados significativamente melhores no que toca às métricas de classificação. Tanto o scheduler como o Optuna aumentavam substancialmente o tempo de treino do modelo, o que contribui-o consideravelmente para a decisão de não utilizar estes. Por fim, no caso específico do Optuna é importante realçar que foram, também, feitas experiências utilizando um timeout entre dez e trinta minutos, com o intuito de reduzir o tempo necessário para o treino do modelo, que chegou a resultar em treinos onde o modelo obtinha resultados piores.

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