In [1]:
```
import tensorflow as tf
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from keras import layers
from keras import models
from tensorflow.keras import optimizers
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import numpy as np
```

```
2025-06-11 23:21:04.520766: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1749680464.534134   15362 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1749680464.538076   15362 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
W0000 00:00:1749680464.547928   15362 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749680464.547944   15362 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749680464.547945   15362 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749680464.547946   15362 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
2025-06-11 23:21:04.551293: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.19.0'
```

MODELO S - Optuna

O presente notebook foi testado, numa segunda abordagem, com as configuraçoes obtidas pelo optuna

1 - Data Preprocessing

Caminhos dos sets

In [3]:
```
train_dir = 'dataset_balanceado_final/train'
validation_dir = 'dataset_balanceado_final/validation'
test_dir = 'dataset_balanceado_final/test'
```

Definir batch_size e image_size

In [4]:
```
from tensorflow.keras.utils import image_dataset_from_directory

IMG_SIZE = 150
BATCH_SIZE = 32
```

Training set - É o conjunto de dados usado para treinar a rede

In [5]:
```
train_dataset = image_dataset_from_directory(
    train_dir,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    label_mode='categorical' # categorical porque temos várias classes, senão se
)
```

```
Found 4276 files belonging to 7 classes.
I0000 00:00:1749680466.967290   15362 gpu_device.cc:2019] Created device /job:loc
alhost/replica:0/task:0/device:GPU:0 with 4804 MB memory:  -> device: 0, name: NV
IDIA GeForce GTX 1660 SUPER, pci bus id: 0000:03:00.0, compute capability: 7.5
I0000 00:00:1749680466.970832   15362 gpu_device.cc:2019] Created device /job:loc
alhost/replica:0/task:0/device:GPU:1 with 4804 MB memory:  -> device: 1, name: NV
IDIA GeForce GTX 1660 SUPER, pci bus id: 0000:05:00.0, compute capability: 7.5
```

Validation set - Usado para 'testar' o modelo durante o processo de procura da melhor combinação de hiperparâmetros.

In [6]:
```
validation_dataset = image_dataset_from_directory(
    validation_dir,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    label_mode='categorical'
)
```

```
Found 1420 files belonging to 7 classes.
```

Test set - Usado para testar o modelo depois do processo de treino

In [7]:
```
test_dataset = image_dataset_from_directory(
    test_dir,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    label_mode='categorical'
)
```

```
Found 1420 files belonging to 7 classes.
```

Data Augmentation

Gerar mais dados de treino a partir de amostras de treino existentes, aumentando as amostras através de uma série de transformações aleatórias que produzem imagens com aspeto credível.

In [8]:
```
data_augmentation = keras.Sequential([layers.RandomFlip("horizontal"), # Efeito
                                      layers.RandomRotation(0.1),
                                      layers.RandomTranslation(0.1, 0.2),
                                      layers.RandomFlip("vertical"), # rotaçao a
                                      layers.RandomZoom(0.2),]) # rotação de 20%
```

Métricas para avaliar os modelos

In [9]:
```
# Utiliza uma função(do sckicit-learn) para avaliar o desempenho do modelo, indi
    # f1-score do modelo
    # accuracy do modelo
    # accuracy por classe

from sklearn.metrics import classification_report
import numpy as np

def print_classification_metrics(model, dataset, phase_name):
    y_true = []
    y_pred = []

    for images, labels in dataset:
        preds = model.predict(images)
        y_true.extend(np.argmax(labels.numpy(), axis=1))
        y_pred.extend(np.argmax(preds, axis=1))

    print(f"\n {phase_name}")
    print(classification_report(y_true, y_pred, digits=4))
```

2 - Construir a CNN (convolucional Neural Network)

In [10]:
```
# Define a camada de entrada do modelo com o formato das imagens (altura, largur
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
```

In [11]:
```
#Aplica as transformações aleatórias às imagens (definidas no inicio do notebook
x = data_augmentation(inputs)
```

In [12]:
```
# Normaliza os valores dos pixels das imagens de entrada para o intervalo [0, 1]
x = layers.Rescaling(1./255)(x)
```

Passo 1 : Camada Convolucional

In [13]:
```
# Primeira camada convolucional com 64 filtros 3x3 e ativação ReLU
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu", padding="same")(
# adicionado o padding="same" para garantir que o output da camada convolucional
```

Passo 2 : Camada de Pooling

In [14]:
```
# Primeira camada de pooling máximo (2x2) para reduzir a dimensionalidade.
x = layers.MaxPooling2D(pool_size=2)(x)
```

Passo 3 : Adicionar mais camadas

In [15]:
```
# Segunda camada convolucional com 128 filtros 3x3 e ativação ReLU.
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding="same")
# Segunda camada de pooling máximo (2x2).
x = layers.MaxPooling2D(pool_size=2)(x)
```

In [16]:
```
# Terceira camada convolucional com 128 filtros 3x3 e ativação ReLU.
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding="same")
# Terceira camada de pooling máximo (2x2).
x = layers.MaxPooling2D(pool_size=2)(x)
```

Passo 4 : Flattening

In [17]:
```
# Achata o output das camadas convolucionais para um vetor 1D.
x = layers.Flatten()(x)
```

BATCH NORMALIZATION (Facultativo)

In [18]:
```
# tirar comentario desta linha abaixo se queremos usar batch normalization
# porem foi testado varias vezes e em diferentes camadas da rede, mas não melhor
#x = layers.BatchNormalization( axis=-1, momentum=0.99, epsilon=0.001, center=Tr
```

Dropout (optuna disse para não usar) - funçao de regularização

In [19]:
```
# Aplica Dropout (50%) para desativar aleatoriamente neurónios, prevenindo o ove
x = layers.Dropout(0.5)(x)
```

Passo 5 : Full Connection

In [20]:
```
# Definir funçao de Regularização L2 dada pelo optuna
reg = regularizers.l2(0.00002426218)  # Executar para ativar

# Camada densa (totalmente conectada) com 128 neurónios, ativação ReLU e funçao
x = layers.Dense(128, activation="relu", kernel_regularizer=reg)(x)
```

Passo 6 : Output Layer

In [21]:
```
# Camada de saída densa com 7 neurónios e ativação Softmax (para classificação c
outputs = layers.Dense(7, activation="softmax")(x)
```

In [22]:
```
# Cria o modelo Keras usando as camadas de entrada e saída definidas.
model = keras.Model(inputs=inputs, outputs=outputs)
```

3 - Treinar a rede CNN

Funções de otimização disponíveis: Adam, RMSprop e SGD

Funções de loss disponíveis: categorical_crossentropy , KLDivergence e MSE

In [23]:
```
# Configura o otimizador Adam com Learning rate de 0.001
# Define a função de loss: Categorical_Crossentropy com label smoothing de 0.095
# Indica que a 'accuracy' (precisão) será a métrica durante o treino.
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.095),
    metrics=['accuracy'])
```

Treinar o modelo

In [ ]:
```
#utilizar early stopping -> o modelo pára de treinar se a accuracy nao subir dur
callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ModelCheckpoint(filepath='models/melhor_modelo.h5', save_best_only=True, mon
]

history = model.fit(
    train_dataset, #Inicia o treino do modelo usando o conjunto de dados de trei
    epochs=100,  # Apesar de ter um numero alto aqui, o EarlyStopping para autom
    validation_data=validation_dataset # Usa o conjunto de dados de validação pa
    #,callbacks=callbacks
)
```

```
Epoch 1/100
I0000 00:00:1749680471.338111   15429 cuda_dnn.cc:529] Loaded cuDNN version 90300
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 15s 81ms/step - accuracy: 0.1748 - loss: 2.0181 - va
l_accuracy: 0.2782 - val_loss: 1.7705
Epoch 2/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.3391 - loss: 1.7492 - va
l_accuracy: 0.4120 - val_loss: 1.5985
Epoch 3/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.4252 - loss: 1.5830 - va
l_accuracy: 0.4106 - val_loss: 1.5637
Epoch 4/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.4626 - loss: 1.5161 - va
l_accuracy: 0.4676 - val_loss: 1.4421
Epoch 5/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5031 - loss: 1.4521 - va
l_accuracy: 0.5035 - val_loss: 1.4243
Epoch 6/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5061 - loss: 1.4401 - va
l_accuracy: 0.5458 - val_loss: 1.3827
Epoch 7/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5218 - loss: 1.3938 - va
l_accuracy: 0.5500 - val_loss: 1.3593
Epoch 8/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5228 - loss: 1.4072 - va
l_accuracy: 0.4768 - val_loss: 1.4665
Epoch 9/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5308 - loss: 1.3875 - va
l_accuracy: 0.5634 - val_loss: 1.3369
Epoch 10/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5318 - loss: 1.3517 - va
l_accuracy: 0.5458 - val_loss: 1.3706
Epoch 11/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5464 - loss: 1.3581 - va
l_accuracy: 0.5606 - val_loss: 1.3318
Epoch 12/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5568 - loss: 1.3248 - va
l_accuracy: 0.5718 - val_loss: 1.3243
Epoch 13/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5522 - loss: 1.3495 - va
l_accuracy: 0.5739 - val_loss: 1.3238
Epoch 14/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5581 - loss: 1.3305 - va
l_accuracy: 0.6021 - val_loss: 1.2958
Epoch 15/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5571 - loss: 1.3373 - va
l_accuracy: 0.5817 - val_loss: 1.2872
Epoch 16/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5643 - loss: 1.3186 - va
l_accuracy: 0.5965 - val_loss: 1.2928
Epoch 17/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5805 - loss: 1.2985 - va
l_accuracy: 0.5979 - val_loss: 1.2779
Epoch 18/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5807 - loss: 1.2909 - va
l_accuracy: 0.5908 - val_loss: 1.2716
Epoch 19/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5873 - loss: 1.2815 - va
l_accuracy: 0.6070 - val_loss: 1.2713
Epoch 20/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5999 - loss: 1.2567 - va
l_accuracy: 0.5824 - val_loss: 1.2820
Epoch 21/100
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5980 - loss: 1.2617 - va
l_accuracy: 0.6042 - val_loss: 1.2639
Epoch 22/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.5928 - loss: 1.2677 - va
l_accuracy: 0.6007 - val_loss: 1.2729
Epoch 23/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6094 - loss: 1.2511 - va
l_accuracy: 0.6049 - val_loss: 1.2426
Epoch 24/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6105 - loss: 1.2400 - va
l_accuracy: 0.6092 - val_loss: 1.2515
Epoch 25/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6257 - loss: 1.2173 - va
l_accuracy: 0.6254 - val_loss: 1.2185
Epoch 26/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6153 - loss: 1.2322 - va
l_accuracy: 0.6275 - val_loss: 1.2234
Epoch 27/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6206 - loss: 1.2268 - va
l_accuracy: 0.6070 - val_loss: 1.2528
Epoch 28/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6235 - loss: 1.2248 - va
l_accuracy: 0.6359 - val_loss: 1.2228
Epoch 29/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6240 - loss: 1.2158 - va
l_accuracy: 0.5993 - val_loss: 1.3083
Epoch 30/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6425 - loss: 1.2015 - va
l_accuracy: 0.6521 - val_loss: 1.1834
Epoch 31/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6386 - loss: 1.2059 - va
l_accuracy: 0.5979 - val_loss: 1.2627
Epoch 32/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6338 - loss: 1.2047 - va
l_accuracy: 0.6465 - val_loss: 1.2058
Epoch 33/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6170 - loss: 1.2219 - va
l_accuracy: 0.6451 - val_loss: 1.2148
Epoch 34/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6321 - loss: 1.2100 - va
l_accuracy: 0.6141 - val_loss: 1.2400
Epoch 35/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6495 - loss: 1.1978 - va
l_accuracy: 0.6394 - val_loss: 1.2274
Epoch 36/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.6363 - loss: 1.1933 - va
l_accuracy: 0.6437 - val_loss: 1.1966
Epoch 37/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6475 - loss: 1.1891 - va
l_accuracy: 0.5965 - val_loss: 1.2918
Epoch 38/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6354 - loss: 1.2030 - va
l_accuracy: 0.6549 - val_loss: 1.1831
Epoch 39/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6594 - loss: 1.1837 - va
l_accuracy: 0.6394 - val_loss: 1.2312
Epoch 40/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6556 - loss: 1.1970 - va
l_accuracy: 0.6648 - val_loss: 1.1855
Epoch 41/100
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6590 - loss: 1.1661 - va
l_accuracy: 0.6599 - val_loss: 1.2110
Epoch 42/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6434 - loss: 1.1821 - va
l_accuracy: 0.6775 - val_loss: 1.1814
Epoch 43/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6592 - loss: 1.1757 - va
l_accuracy: 0.6838 - val_loss: 1.1587
Epoch 44/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6682 - loss: 1.1512 - va
l_accuracy: 0.6655 - val_loss: 1.2150
Epoch 45/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6679 - loss: 1.1551 - va
l_accuracy: 0.6500 - val_loss: 1.2279
Epoch 46/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6791 - loss: 1.1419 - va
l_accuracy: 0.6739 - val_loss: 1.1616
Epoch 47/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6695 - loss: 1.1627 - va
l_accuracy: 0.6887 - val_loss: 1.1709
Epoch 48/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6564 - loss: 1.1748 - va
l_accuracy: 0.6725 - val_loss: 1.1905
Epoch 49/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6796 - loss: 1.1693 - va
l_accuracy: 0.6683 - val_loss: 1.2044
Epoch 50/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6880 - loss: 1.1456 - va
l_accuracy: 0.6521 - val_loss: 1.2240
Epoch 51/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6753 - loss: 1.1631 - va
l_accuracy: 0.6930 - val_loss: 1.1706
Epoch 52/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6605 - loss: 1.1686 - va
l_accuracy: 0.6908 - val_loss: 1.1512
Epoch 53/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6930 - loss: 1.1304 - va
l_accuracy: 0.6754 - val_loss: 1.1759
Epoch 54/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6862 - loss: 1.1500 - va
l_accuracy: 0.6718 - val_loss: 1.1785
Epoch 55/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6954 - loss: 1.1204 - va
l_accuracy: 0.6887 - val_loss: 1.1691
Epoch 56/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6879 - loss: 1.1361 - va
l_accuracy: 0.6746 - val_loss: 1.1915
Epoch 57/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6845 - loss: 1.1407 - va
l_accuracy: 0.6796 - val_loss: 1.1783
Epoch 58/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7066 - loss: 1.1148 - va
l_accuracy: 0.6930 - val_loss: 1.1503
Epoch 59/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7081 - loss: 1.1269 - va
l_accuracy: 0.6993 - val_loss: 1.1605
Epoch 60/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6917 - loss: 1.1193 - va
l_accuracy: 0.6831 - val_loss: 1.1800
Epoch 61/100
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6855 - loss: 1.1436 - va
l_accuracy: 0.6887 - val_loss: 1.1632
Epoch 62/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7031 - loss: 1.1100 - va
l_accuracy: 0.6930 - val_loss: 1.1473
Epoch 63/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7150 - loss: 1.1021 - va
l_accuracy: 0.6662 - val_loss: 1.2047
Epoch 64/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7134 - loss: 1.1109 - va
l_accuracy: 0.6944 - val_loss: 1.1584
Epoch 65/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7119 - loss: 1.1157 - va
l_accuracy: 0.6782 - val_loss: 1.1821
Epoch 66/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7087 - loss: 1.1146 - va
l_accuracy: 0.6901 - val_loss: 1.1649
Epoch 67/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7119 - loss: 1.1010 - va
l_accuracy: 0.6915 - val_loss: 1.1663
Epoch 68/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7026 - loss: 1.1073 - va
l_accuracy: 0.7092 - val_loss: 1.1285
Epoch 69/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7089 - loss: 1.1034 - va
l_accuracy: 0.6704 - val_loss: 1.2017
Epoch 70/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7046 - loss: 1.1256 - va
l_accuracy: 0.7141 - val_loss: 1.1351
Epoch 71/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7121 - loss: 1.1014 - va
l_accuracy: 0.7148 - val_loss: 1.1266
Epoch 72/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.6988 - loss: 1.1108 - va
l_accuracy: 0.7092 - val_loss: 1.1421
Epoch 73/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7202 - loss: 1.1019 - va
l_accuracy: 0.6894 - val_loss: 1.1971
Epoch 74/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7127 - loss: 1.1073 - va
l_accuracy: 0.6923 - val_loss: 1.1692
Epoch 75/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7108 - loss: 1.1019 - va
l_accuracy: 0.7085 - val_loss: 1.1196
Epoch 76/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7009 - loss: 1.0935 - va
l_accuracy: 0.6915 - val_loss: 1.1916
Epoch 77/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7150 - loss: 1.0960 - va
l_accuracy: 0.7113 - val_loss: 1.1245
Epoch 78/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7117 - loss: 1.0911 - va
l_accuracy: 0.6880 - val_loss: 1.1849
Epoch 79/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7095 - loss: 1.0970 - va
l_accuracy: 0.7007 - val_loss: 1.1471
Epoch 80/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7226 - loss: 1.0791 - va
l_accuracy: 0.7049 - val_loss: 1.1348
Epoch 81/100
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7356 - loss: 1.0672 - va
l_accuracy: 0.7070 - val_loss: 1.1377
Epoch 82/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7312 - loss: 1.0692 - va
l_accuracy: 0.6704 - val_loss: 1.1812
Epoch 83/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7281 - loss: 1.0852 - va
l_accuracy: 0.6894 - val_loss: 1.1922
Epoch 84/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7324 - loss: 1.0706 - va
l_accuracy: 0.6845 - val_loss: 1.1902
Epoch 85/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7136 - loss: 1.0872 - va
l_accuracy: 0.7014 - val_loss: 1.1579
Epoch 86/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7259 - loss: 1.0738 - va
l_accuracy: 0.7183 - val_loss: 1.1384
Epoch 87/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7328 - loss: 1.0762 - va
l_accuracy: 0.7070 - val_loss: 1.1374
Epoch 88/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7316 - loss: 1.0678 - va
l_accuracy: 0.7021 - val_loss: 1.1637
Epoch 89/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7241 - loss: 1.0811 - va
l_accuracy: 0.7148 - val_loss: 1.1379
Epoch 90/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7211 - loss: 1.0976 - va
l_accuracy: 0.6852 - val_loss: 1.1960
Epoch 91/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7287 - loss: 1.0831 - va
l_accuracy: 0.7021 - val_loss: 1.1529
Epoch 92/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7253 - loss: 1.0875 - va
l_accuracy: 0.6845 - val_loss: 1.1971
Epoch 93/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7259 - loss: 1.0765 - va
l_accuracy: 0.7113 - val_loss: 1.1364
Epoch 94/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7251 - loss: 1.0769 - va
l_accuracy: 0.7063 - val_loss: 1.1537
Epoch 95/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7413 - loss: 1.0630 - va
l_accuracy: 0.6930 - val_loss: 1.1774
Epoch 96/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7440 - loss: 1.0644 - va
l_accuracy: 0.7134 - val_loss: 1.1474
Epoch 97/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7258 - loss: 1.0715 - va
l_accuracy: 0.7014 - val_loss: 1.1705
Epoch 98/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7459 - loss: 1.0593 - va
l_accuracy: 0.7000 - val_loss: 1.1623
Epoch 99/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7518 - loss: 1.0428 - va
l_accuracy: 0.7014 - val_loss: 1.1733
Epoch 100/100
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.7338 - loss: 1.0658 - va
l_accuracy: 0.7021 - val_loss: 1.1522
```

4 - Testar o modelo

In [25]:
```
print_classification_metrics(model, test_dataset, "Modelo 1 : CNN de raiz")
```

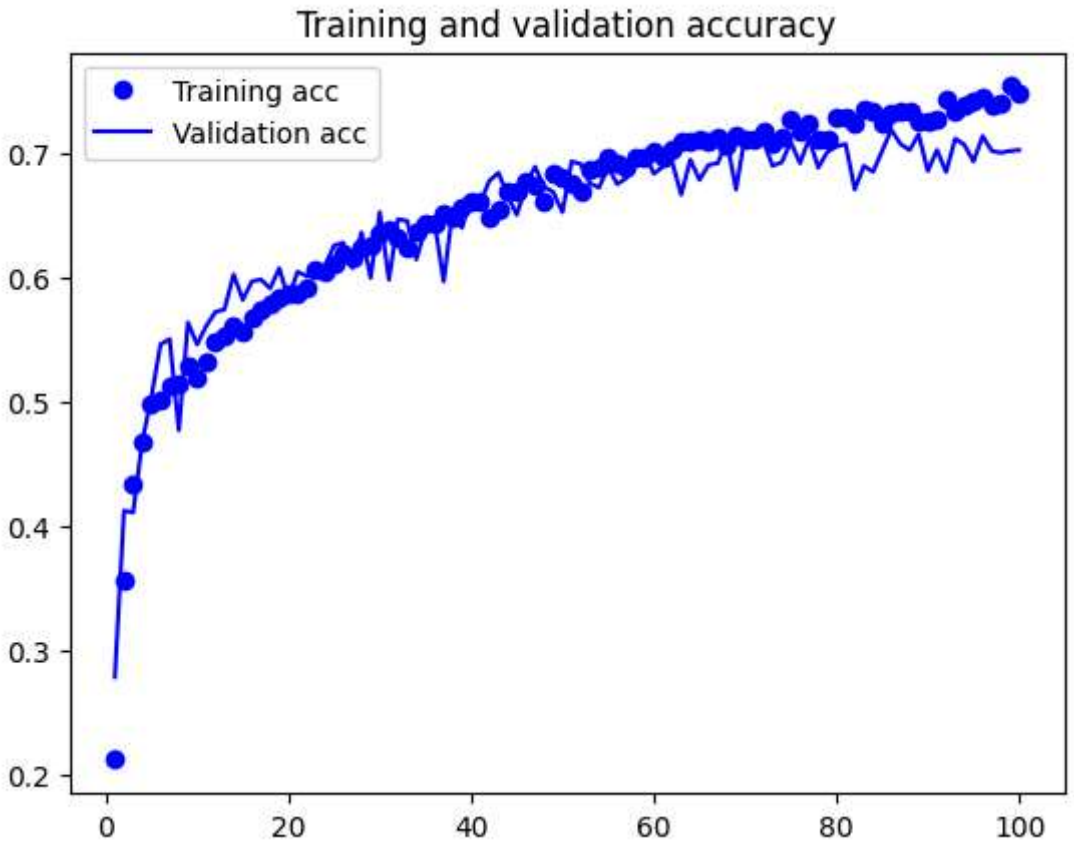
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step

 Modelo 1 : CNN de raiz
              precision    recall  f1-score   support

           0     0.5694    0.6308    0.5985       195
           1     0.6923    0.5455    0.6102       198
           2     0.6172    0.5890    0.6028       219
           3     0.7336    0.8750    0.7981       192
           4     0.5826    0.6351    0.6078       222
           5     0.7733    0.6650    0.7151       200
           6     0.9592    0.9691    0.9641       194

    accuracy                         0.6972      1420
   macro avg     0.7040    0.7014    0.6995      1420
```

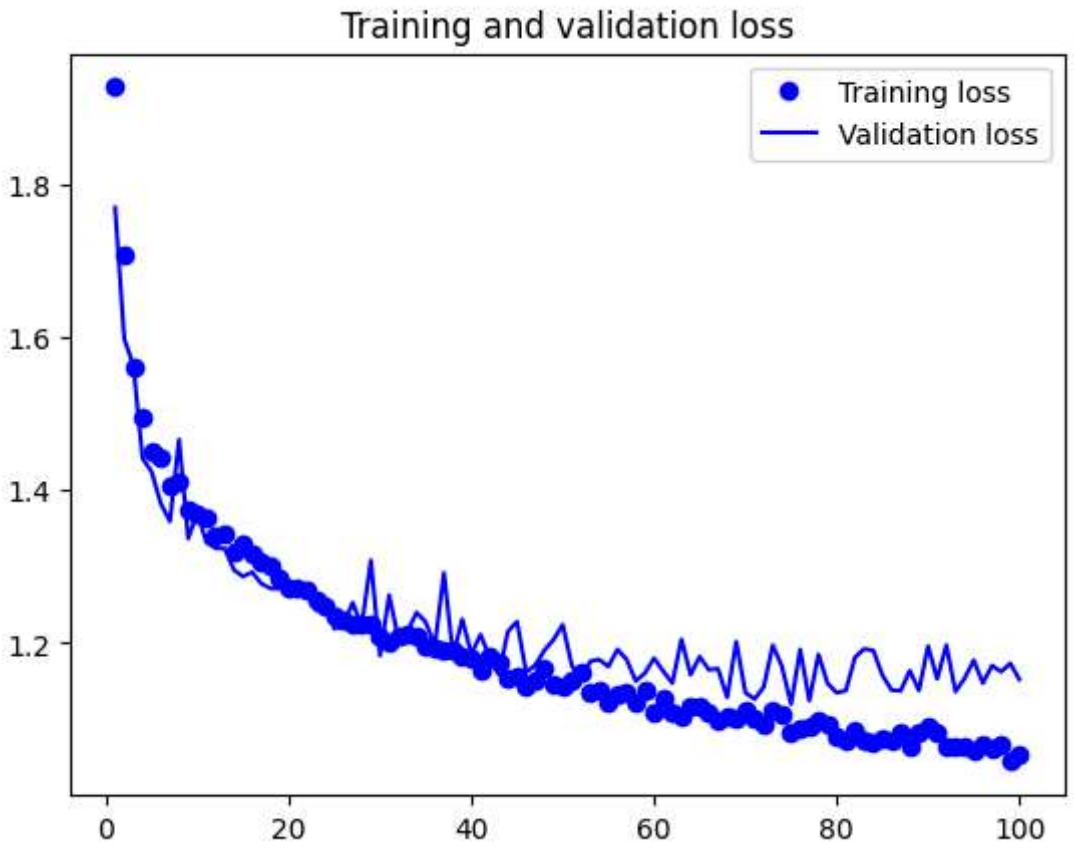
```
weighted avg      0.7002    0.6972    0.6956       1420
```

```
2025-06-11 23:38:11.586496: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Curvas de Loss e de Accuracy

In [26]:
```
import matplotlib.pyplot as plt
accuracy = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





A acurácia de treino e validação aumentam de forma consistente ao longo das épocas, com valores próximos e sem grande divergência, o que indica um bom ajuste do modelo sem sinais claros de overfitting.

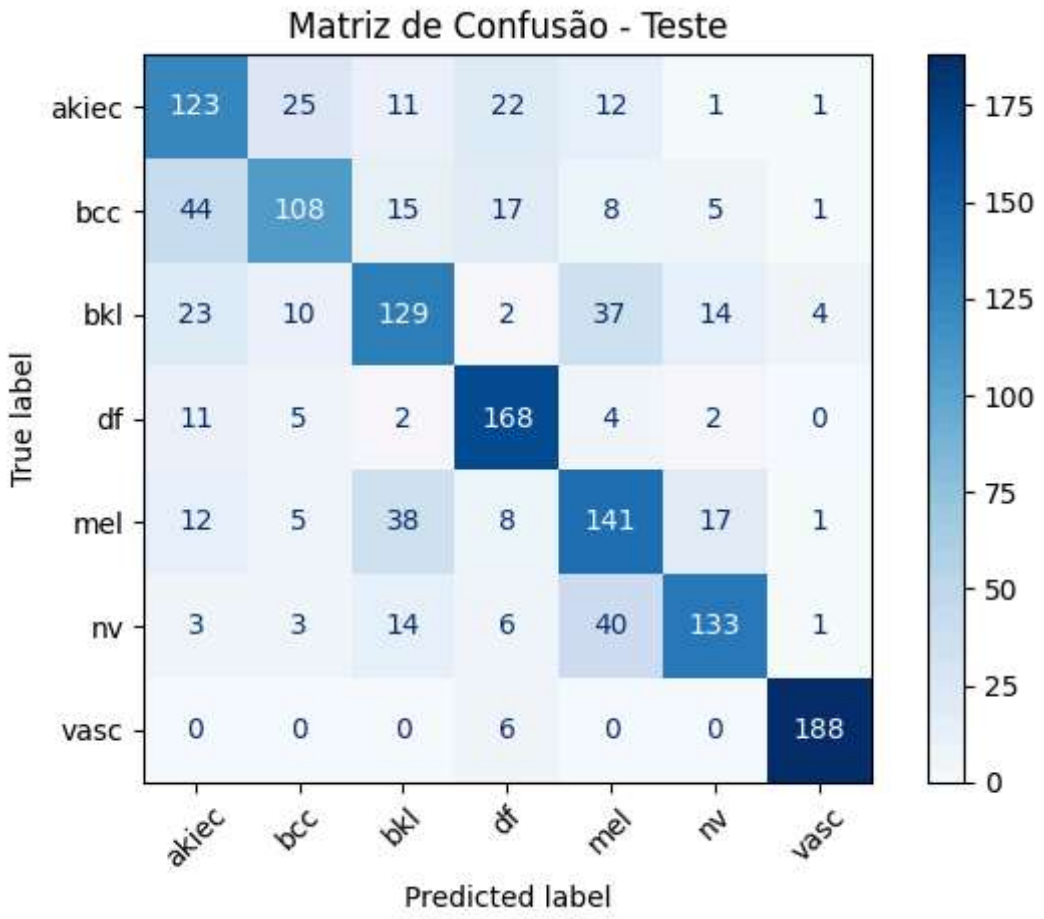
As curvas de loss de treino e validação diminuem gradualmente, mantendo-se próximas e estáveis ao longo do tempo. Isso mostra que o modelo está a aprender de forma eficiente e generaliza bem para os dados de validação.

Matriz de Confusão

In [27]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Obter predições no test_dataset
y_true = []
y_pred = []

for images, labels in test_dataset:
    preds = model.predict(images)
    y_true.extend(np.argmax(labels.numpy(), axis=1))
    y_pred.extend(np.argmax(preds, axis=1))

class_names = test_dataset.class_names

# Criar e mostrar a matriz de confusão
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

plt.figure(figsize=(10, 8))
```

```
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45, values_format='d')
plt.title("Matriz de Confusão - Teste")
plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
2025-06-11 23:38:15.473451: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
<Figure size 1000x800 with 0 Axes>
```



O modelo apresenta um bom desempenho global, com elevada taxa de acerto nas classes 'df', 'nv' e 'vasc'. No entanto, observa-se confusão significativa entre 'akiec', 'bcc' e 'bkl', o que pode dever-se à semelhança visual entre estas classes.

Salvar o modelo

In [ ]:
```
model.save("modelS_2C_optuna_com_data_aug_adam_cat_cross.keras")
```

```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failur
e.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```