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
import numpy as np
```

```
2025-06-10 22:14:26.459062: 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 00:00:1749590066.472617 3052959 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1749590066.476775 3052959 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
W0000 00:00:1749590066.487264 3052959 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749590066.487282 3052959 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749590066.487283 3052959 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
W0000 00:00:1749590066.487284 3052959 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than once.
2025-06-10 22:14:26.490784: 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

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 00:00:1749590068.836593 3052959 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 00:00:1749590068.840103 3052959 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 [ ]:
```
data_augmentation = keras.Sequential([layers.RandomFlip("horizontal"), # Efeito
                                      layers.RandomRotation(0.1), # rotaçao de 1
                                      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 (facultativo) - 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 (opcional)
reg = regularizers.l2(0.01)  # 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
# Define a função de loss: KLDivergence
# Indica que a 'accuracy' (precisão) será a métrica durante o treino.
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.KLDivergence(),
    metrics=['accuracy'])
```

In [24]:
```
#model.compile(
    #optimizer='SGD',
    #loss='mse',
    #metrics=['accuracy'])
```

In [25]:
```
#model.compile(
    #optimizer=tf.keras.optimizers.RMSprop(),
    #loss=tf.keras.losses.KLDivergence(),
    #metrics=['accuracy'])
```

Treinar o modelo

In [26]:
```
history = model.fit(
    train_dataset, #Inicia o treino do modelo usando o conjunto de dados de trei
    epochs=25,     # O modelo será treinado por 25 épocas (passagens completas p
    validation_data=validation_dataset) # Usa o conjunto de dados de validação p
```

```
Epoch 1/25
I0000 00:00:1749590072.986723 3053018 cuda_dnn.cc:529] Loaded cuDNN version 90300
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 14s 80ms/step - accuracy: 0.1804 - loss: 2.9115 - va
l_accuracy: 0.2507 - val_loss: 1.9783
Epoch 2/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.3034 - loss: 1.9019 - va
l_accuracy: 0.3254 - val_loss: 1.8228
Epoch 3/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.3868 - loss: 1.7169 - va
l_accuracy: 0.4789 - val_loss: 1.5583
Epoch 4/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4316 - loss: 1.5989 - va
l_accuracy: 0.5035 - val_loss: 1.4842
Epoch 5/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4726 - loss: 1.4959 - va
l_accuracy: 0.4951 - val_loss: 1.4498
Epoch 6/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4765 - loss: 1.4581 - va
l_accuracy: 0.4831 - val_loss: 1.5025
Epoch 7/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4846 - loss: 1.4575 - va
l_accuracy: 0.4789 - val_loss: 1.5565
Epoch 8/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4989 - loss: 1.4091 - va
l_accuracy: 0.5394 - val_loss: 1.3455
Epoch 9/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5071 - loss: 1.4035 - va
l_accuracy: 0.4887 - val_loss: 1.4702
Epoch 10/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.4975 - loss: 1.4033 - va
l_accuracy: 0.5458 - val_loss: 1.3391
Epoch 11/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5175 - loss: 1.3452 - va
l_accuracy: 0.5141 - val_loss: 1.4211
Epoch 12/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5057 - loss: 1.3954 - va
l_accuracy: 0.5070 - val_loss: 1.4471
Epoch 13/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5250 - loss: 1.3703 - va
l_accuracy: 0.5479 - val_loss: 1.3006
Epoch 14/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5237 - loss: 1.3504 - va
l_accuracy: 0.5099 - val_loss: 1.4027
Epoch 15/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5286 - loss: 1.3746 - va
l_accuracy: 0.5549 - val_loss: 1.3289
Epoch 16/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5208 - loss: 1.3788 - va
l_accuracy: 0.5493 - val_loss: 1.2961
Epoch 17/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5367 - loss: 1.3235 - va
l_accuracy: 0.5465 - val_loss: 1.3486
Epoch 18/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5250 - loss: 1.3372 - va
l_accuracy: 0.5387 - val_loss: 1.3425
Epoch 19/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5391 - loss: 1.3248 - va
l_accuracy: 0.5563 - val_loss: 1.3054
Epoch 20/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5539 - loss: 1.2846 - va
l_accuracy: 0.5479 - val_loss: 1.3432
Epoch 21/25
```

```
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5411 - loss: 1.3123 - va
l_accuracy: 0.5648 - val_loss: 1.3016
Epoch 22/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5298 - loss: 1.3281 - va
l_accuracy: 0.5514 - val_loss: 1.3078
Epoch 23/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5483 - loss: 1.3064 - va
l_accuracy: 0.5592 - val_loss: 1.3390
Epoch 24/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5290 - loss: 1.3327 - va
l_accuracy: 0.5535 - val_loss: 1.3448
Epoch 25/25
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.5280 - loss: 1.3284 - va
l_accuracy: 0.5697 - val_loss: 1.2698
```

4 - Testar o modelo

In [27]:
```
print_classification_metrics(model, test_dataset, "Modelo 1 : CNN de raiz")
```

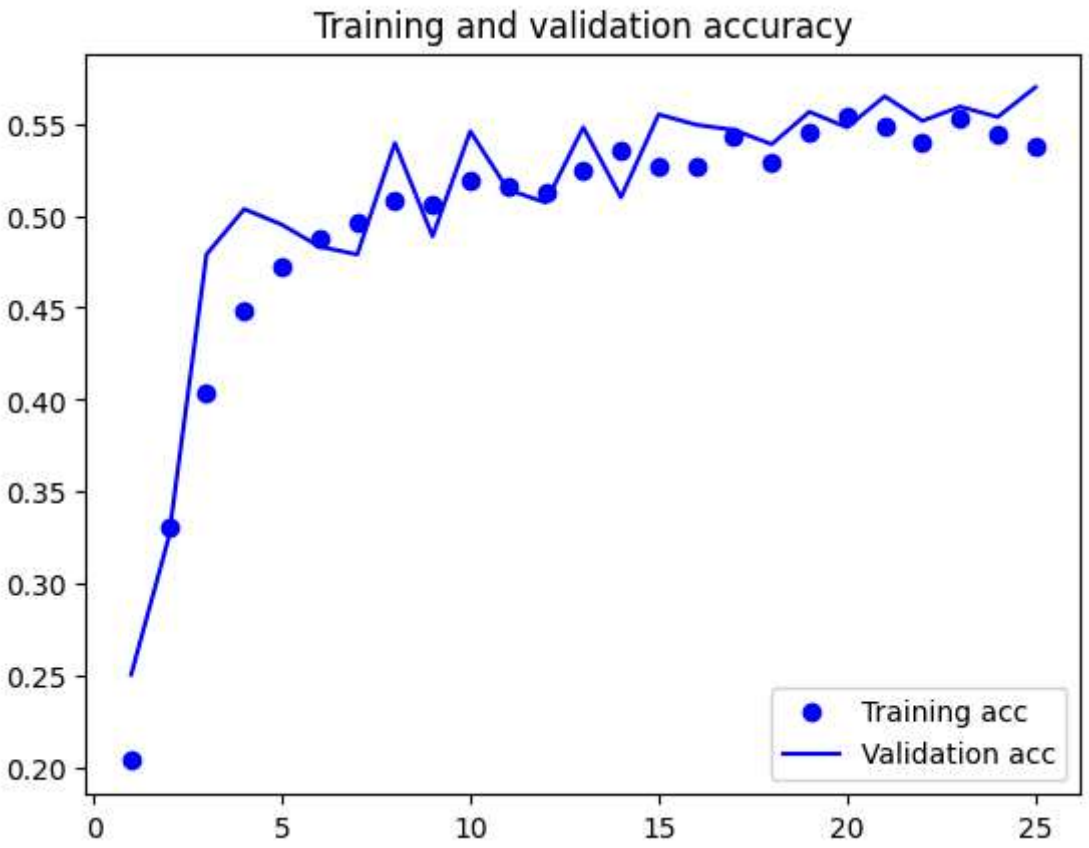
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/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 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/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 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 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step

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

           0     0.4602    0.4154    0.4367       195
           1     0.4839    0.4545    0.4688       198
           2     0.5299    0.6484    0.5832       219
           3     0.5172    0.4688    0.4918       192
           4     0.5311    0.5000    0.5151       222
           5     0.6738    0.6300    0.6512       200
           6     0.8455    0.9588    0.8986       194

    accuracy                         0.5817      1420
   macro avg     0.5774    0.5823    0.5779      1420
```

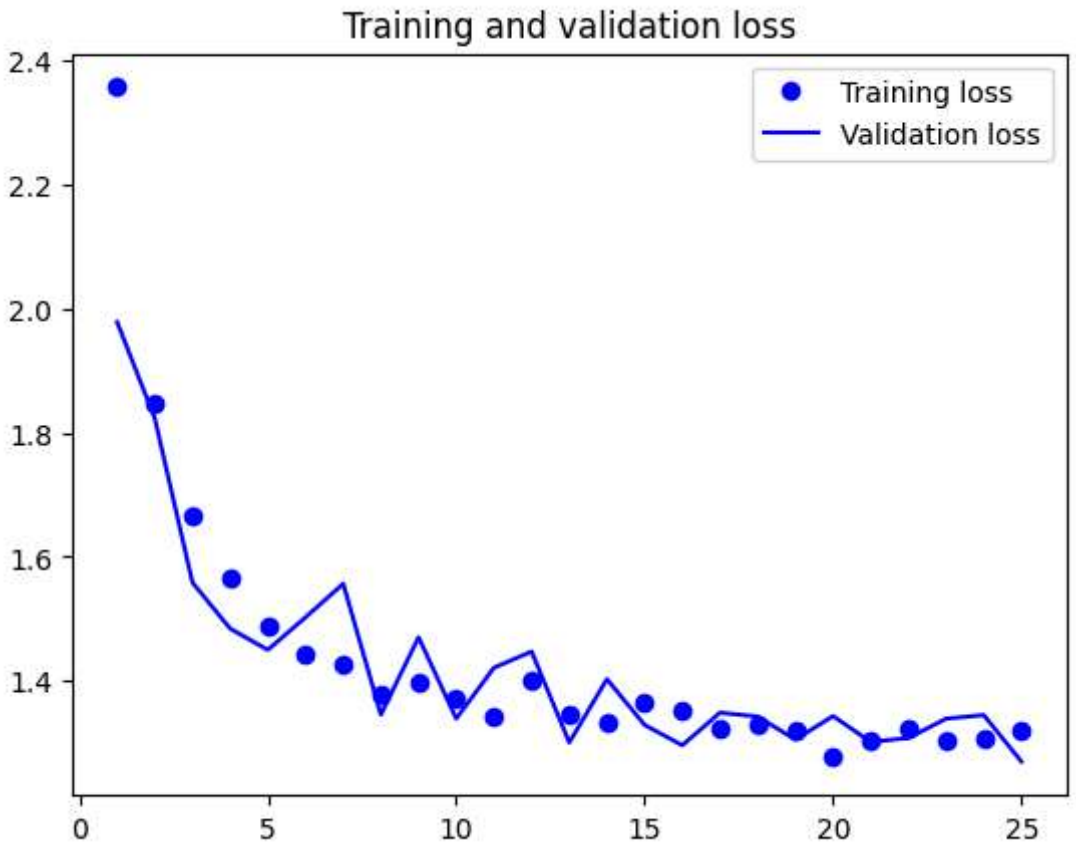
```
weighted avg      0.5758    0.5817    0.5768       1420
```

```
2025-06-10 22:18:47.909411: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Ver as Curvas de Loss e de Accuracy

In [28]:
```
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()
```

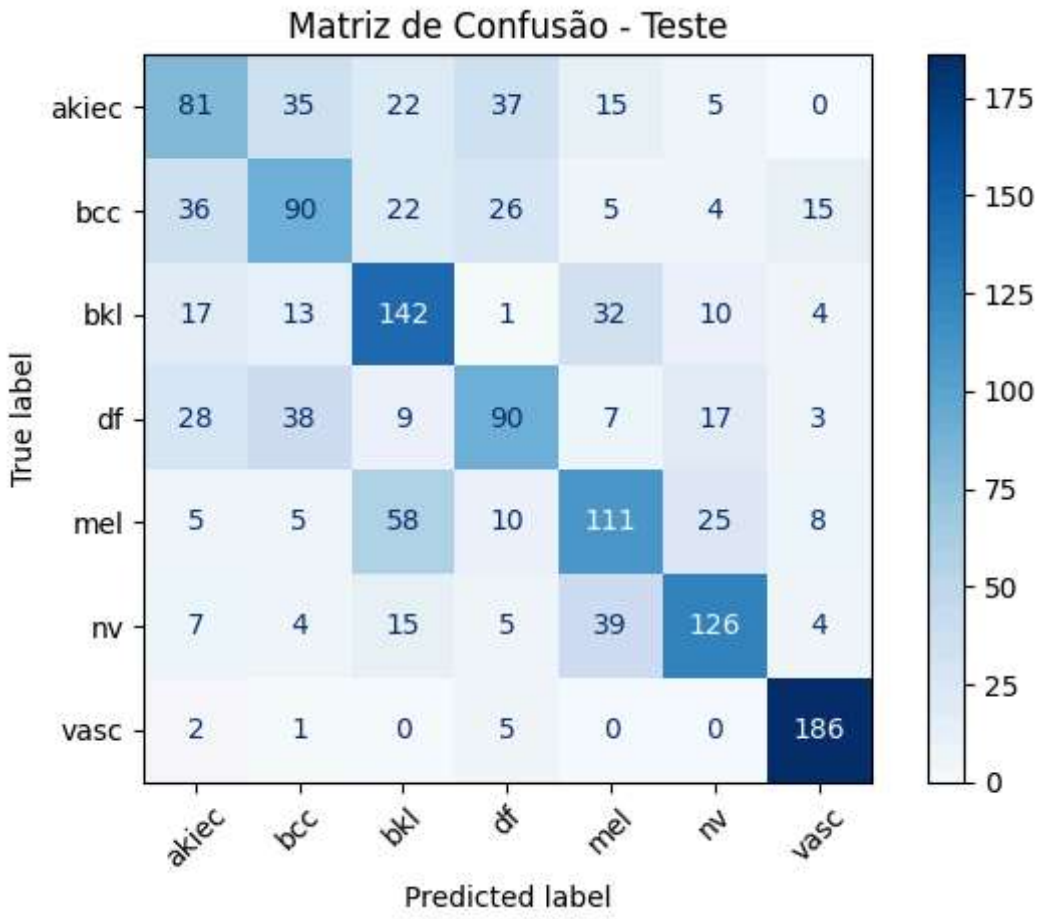




Análise geral dos gráficos acima: O modelo treinou de forma "equilibrada", com desempenho consistente em treino e validação. A estabilidade das curvas sugere que o número de épocas foi adequado e o modelo está bem ajustado.

In [ ]:
```
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 89ms/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 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 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 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 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
2025-06-10 22:48:54.899190: 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>
```




```
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.
```

A matriz de confusão mostra melhor desempenho geral, com classes "vasc", "bkl" e "nv" muito bem classificadas. Algumas confusões persistem entre lesões visualmente semelhantes, como "akiec" e "bcc"

Salvar o modelo

In [29]:
```
model.save("modelS_2B1_com_data_aug_adam_KLD.keras")
```