In [41]:
```
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
```

In [42]:
```
tf.__version__
```

Out[42]:
```
'2.19.0'
```

MODELO S

1 - Data Preprocessing

Caminhos dos sets

In [43]:
```
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 [44]:
```
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 [45]:
```
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.
```

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

In [46]:
```
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 [47]:
```
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 [48]:
```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),          # Espelo horizontal
    layers.RandomRotation(0.1),               # Rotação até 10%
    layers.RandomTranslation(0.1, 0.2),       # "Deslocamento" na largura (10%)
    layers.RandomFlip("vertical"),            # Espelho vertical
    layers.RandomZoom(0.2),                   # Zoom até 20%
])
```

Métricas para avaliar os modelos

In [49]:
```
# 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 [50]:
```
# Define a camada de entrada do modelo com o formato das imagens (altura, largur
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
```

In [51]:
```
#Aplica as transformações aleatórias às imagens (definidas no inicio do notebook
x = data_augmentation(inputs)
```

In [52]:
```
# 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 [53]:
```
# 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 [54]:
```
# Primeira camada de pooling máximo (2x2) para reduzir a dimensionalidade.
x = layers.MaxPooling2D(pool_size=2)(x)
```

Passo 3 : Adicionar mais camadas

In [55]:
```
# 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 [56]:
```
# 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 [57]:
```
# Achata o output das camadas convolucionais para um vetor 1D.
x = layers.Flatten()(x)
```

BATCH NORMALIZATION (Facultativo)

In [58]:
```
# 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 [59]:
```
# Aplica Dropout (50%) para desativar aleatoriamente neurónios, prevenindo o ove
x = layers.Dropout(0.5)(x)
```

Passo 5 : Full Connection

In [60]:
```
# Definir funçao de Regularização L2 (opcional)
#reg = regularizers.l2(0.02)  # Executar para ativar
```

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

Passo 6 : Output Layer

In [61]:
```
# Camada de saída densa com 7 neurónios e ativação Softmax (para classificação c
outputs = layers.Dense(7, activation="softmax")(x)
```

In [62]:
```
# 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 [63]:
```
# Configura o otimizador RMSprop
# Define a função de loss: Categorical Crossentropy
# Indica que a 'accuracy' (precisão) será a métrica durante o treino.
model.compile(
    optimizer=tf.keras.optimizers.RMSprop(),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy'])
```

In [64]:
```
#model.compile(
    #optimizer='SGD',
    #loss='mse',
    #metrics=['accuracy'])
```

In [65]:
```
#model.compile(
    #optimizer=tf.keras.optimizers.RMSprop(),
    #loss=tf.keras.losses.KLDivergence(),
    #metrics=['accuracy'])
```

Treinar o modelo

In [66]:
```
history = model.fit(
    train_dataset, #Inicia o treino do modelo usando o conjunto de dados de trei
    epochs=40,     # O modelo será treinado por 25 épocas (passagens completas p
    validation_data=validation_dataset) # Usa o conjunto de dados de validação p
```

```
Epoch 1/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 11s 74ms/step - accuracy: 0.1362 - loss: 2.5479 - va
l_accuracy: 0.1866 - val_loss: 1.9225
Epoch 2/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.2745 - loss: 1.8447 - va
l_accuracy: 0.4211 - val_loss: 1.5371
Epoch 3/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.4027 - loss: 1.5495 - va
l_accuracy: 0.4570 - val_loss: 1.3830
Epoch 4/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.4429 - loss: 1.4531 - va
l_accuracy: 0.3915 - val_loss: 1.6893
Epoch 5/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.4482 - loss: 1.4223 - va
l_accuracy: 0.4641 - val_loss: 1.4525
Epoch 6/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.4627 - loss: 1.3670 - va
l_accuracy: 0.4556 - val_loss: 1.4431
Epoch 7/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.4853 - loss: 1.3525 - va
l_accuracy: 0.5317 - val_loss: 1.2829
Epoch 8/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5048 - loss: 1.2900 - va
l_accuracy: 0.5444 - val_loss: 1.2128
Epoch 9/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5059 - loss: 1.2810 - va
l_accuracy: 0.5021 - val_loss: 1.3275
Epoch 10/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5064 - loss: 1.2431 - va
l_accuracy: 0.5380 - val_loss: 1.1964
Epoch 11/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5256 - loss: 1.2171 - va
l_accuracy: 0.5099 - val_loss: 1.2289
Epoch 12/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5284 - loss: 1.2166 - va
l_accuracy: 0.5669 - val_loss: 1.1296
Epoch 13/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5370 - loss: 1.2022 - va
l_accuracy: 0.5662 - val_loss: 1.2027
Epoch 14/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5322 - loss: 1.1839 - va
l_accuracy: 0.5782 - val_loss: 1.1437
Epoch 15/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5459 - loss: 1.1726 - va
l_accuracy: 0.4845 - val_loss: 1.4671
Epoch 16/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5597 - loss: 1.1512 - va
l_accuracy: 0.5718 - val_loss: 1.1345
Epoch 17/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5641 - loss: 1.1370 - va
l_accuracy: 0.5331 - val_loss: 1.2868
Epoch 18/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5635 - loss: 1.1200 - va
l_accuracy: 0.5049 - val_loss: 1.2926
Epoch 19/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5881 - loss: 1.0855 - va
l_accuracy: 0.5739 - val_loss: 1.1574
Epoch 20/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5785 - loss: 1.1033 - va
l_accuracy: 0.5239 - val_loss: 1.3362
```

```
Epoch 21/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5574 - loss: 1.1093 - va
l_accuracy: 0.5761 - val_loss: 1.0985
Epoch 22/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5615 - loss: 1.0964 - va
l_accuracy: 0.5690 - val_loss: 1.1269
Epoch 23/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5923 - loss: 1.0724 - va
l_accuracy: 0.5845 - val_loss: 1.1250
Epoch 24/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5773 - loss: 1.0754 - va
l_accuracy: 0.4718 - val_loss: 1.5099
Epoch 25/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5689 - loss: 1.1051 - va
l_accuracy: 0.6085 - val_loss: 1.0514
Epoch 26/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5940 - loss: 1.0433 - va
l_accuracy: 0.5789 - val_loss: 1.0846
Epoch 27/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.5960 - loss: 1.0460 - va
l_accuracy: 0.6162 - val_loss: 1.0597
Epoch 28/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.6050 - loss: 1.0179 - va
l_accuracy: 0.5880 - val_loss: 1.0783
Epoch 29/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.6043 - loss: 1.0359 - va
l_accuracy: 0.5725 - val_loss: 1.1235
Epoch 30/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.6086 - loss: 1.0010 - va
l_accuracy: 0.5606 - val_loss: 1.2484
Epoch 31/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6072 - loss: 0.9964 - va
l_accuracy: 0.6106 - val_loss: 1.0535
Epoch 32/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6282 - loss: 0.9844 - va
l_accuracy: 0.5761 - val_loss: 1.1866
Epoch 33/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6106 - loss: 0.9995 - va
l_accuracy: 0.6049 - val_loss: 1.1496
Epoch 34/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6177 - loss: 0.9848 - va
l_accuracy: 0.6423 - val_loss: 0.9143
Epoch 35/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6203 - loss: 0.9699 - va
l_accuracy: 0.5901 - val_loss: 1.1821
Epoch 36/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6246 - loss: 0.9900 - va
l_accuracy: 0.4739 - val_loss: 1.5944
Epoch 37/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6162 - loss: 0.9982 - va
l_accuracy: 0.6500 - val_loss: 0.9368
Epoch 38/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.6380 - loss: 0.9549 - va
l_accuracy: 0.6162 - val_loss: 1.0826
Epoch 39/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6314 - loss: 0.9668 - va
l_accuracy: 0.6493 - val_loss: 0.9853
Epoch 40/40
134/134 ━━━━━━━━━━━━━━━━━━━━ 10s 74ms/step - accuracy: 0.6546 - loss: 0.9301 - va
l_accuracy: 0.6268 - val_loss: 0.9790
```

4 - Testar o modelo

In [67]:
```
print_classification_metrics(model, test_dataset, "Modelo 1 : CNN de raiz")
```

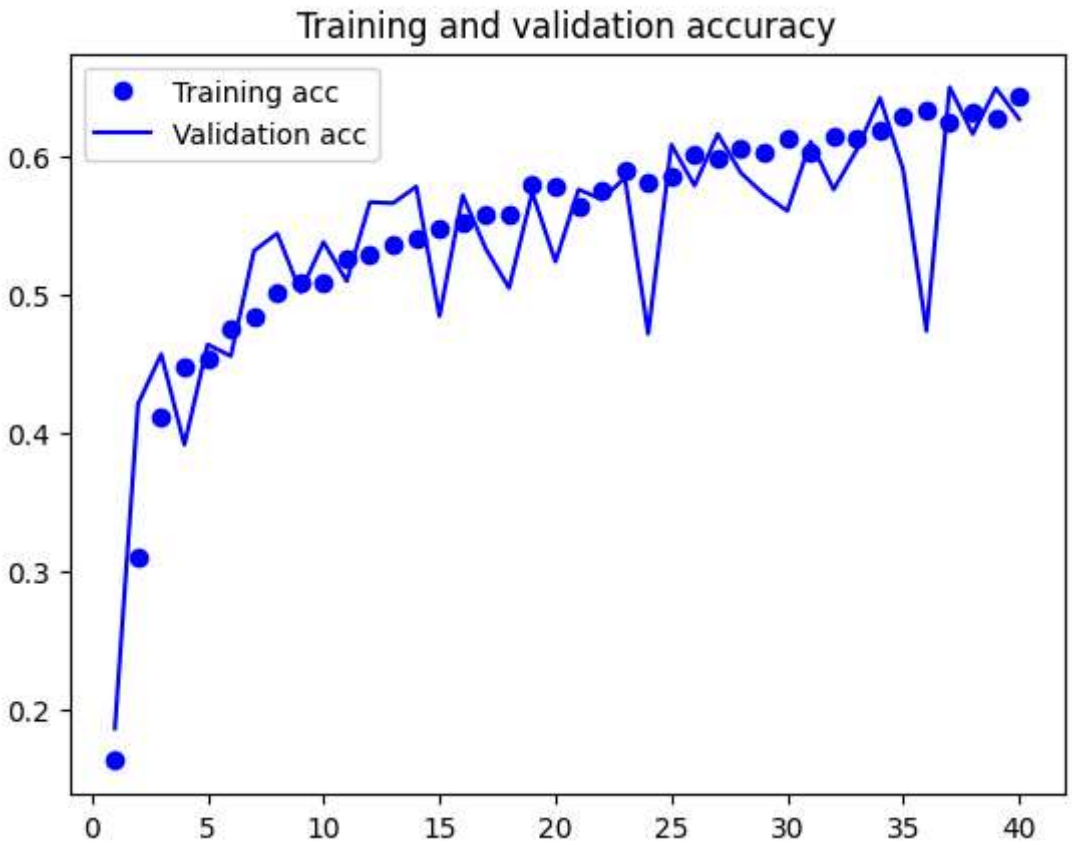
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/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 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/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 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/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 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/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 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step

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

           0     0.4716    0.7231    0.5709       195
           1     0.6456    0.2576    0.3682       198
           2     0.6620    0.4292    0.5208       219
           3     0.5585    0.7708    0.6477       192
           4     0.5633    0.6216    0.5910       222
           5     0.6990    0.6850    0.6919       200
           6     0.9536    0.9536    0.9536       194

    accuracy                         0.6296      1420
   macro avg     0.6505    0.6344    0.6206      1420
```

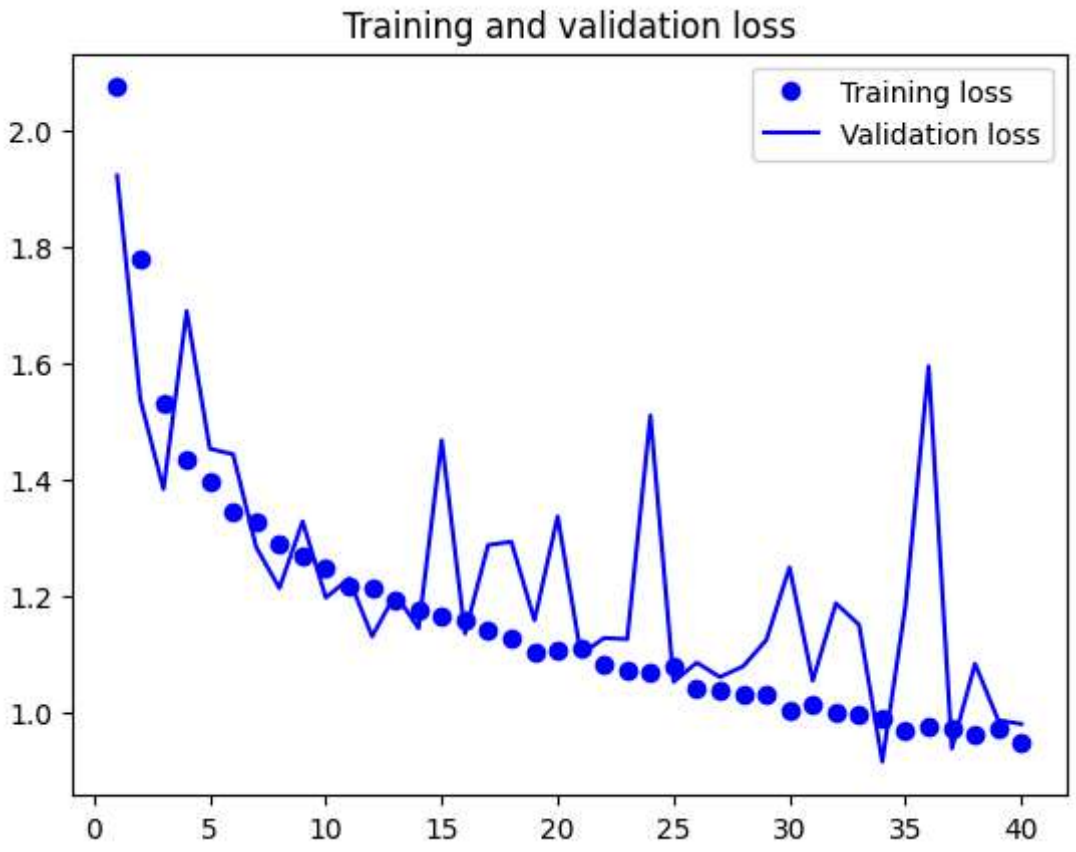
```
weighted avg      0.6492    0.6296    0.6178       1420
```

```
2025-06-11 21:08:26.700624: 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 [72]:
```
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()
```



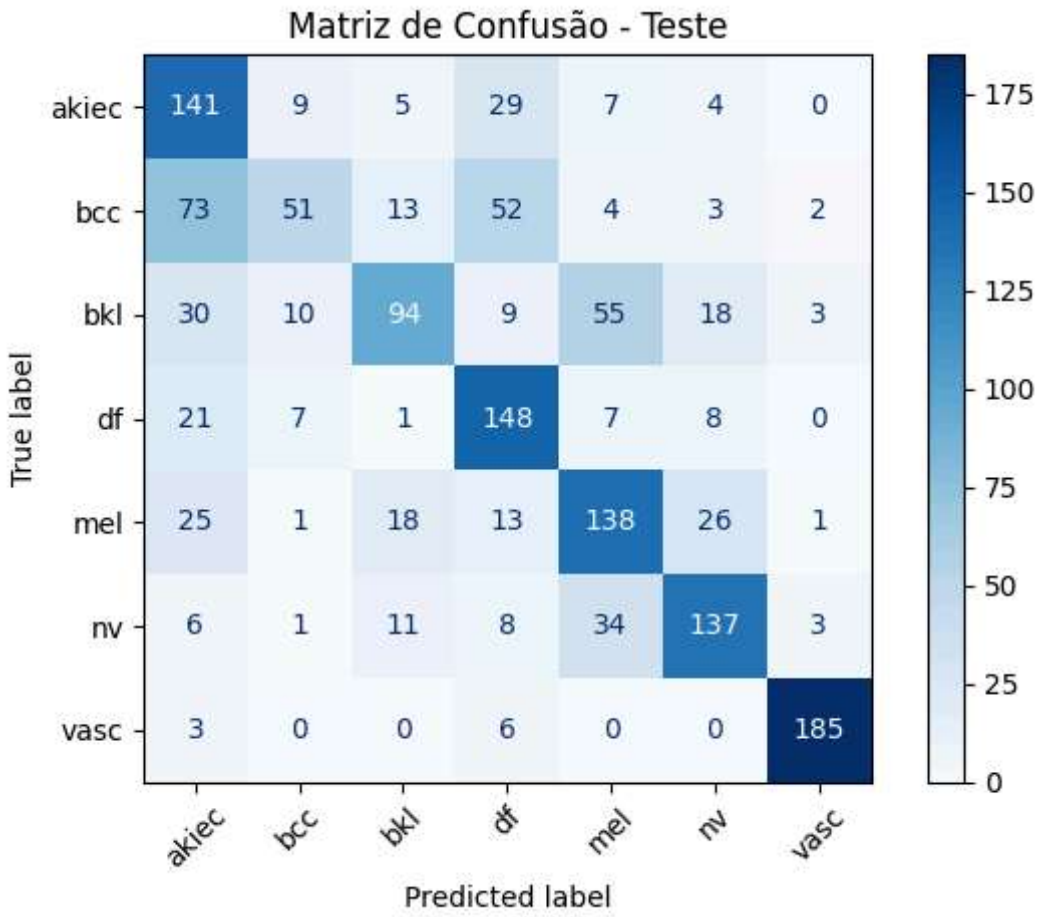


As curvas mostram que o modelo foi bem treinado, com melhoria consistente na acurácia e diminuição contínua da loss. Não há sinais de overfitting. A generalização é boa e o desempenho pode possivelmente ser melhorado com mais épocas ou ajustes finos.

Matriz de Confusão

In [71]:
```
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 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/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 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/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 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 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 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 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
<Figure size 1000x800 with 0 Axes>
```



Salvar o modelo

In [73]:
```
model.save("modelS_2B2_com_data_aug_RMS_cat_cross_best_acc.keras")
```