```python
import numpy as np
import tensorflow as keras
from tensorflow.keras import layers, Model, Input
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
from tensorflow import keras

# Diretórios de dados
base_dir = 'Imagens/'

train_dir = os.path.join(base_dir, 'train/train5')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

Data Augmentation

- Esta técnica é bastante forte contra overfitting pois é um aumento de dados artificial com rotações, zoom, contraste, etc.., o que expõe o modelo a uma variedade maior de exemplos durante o treino, o que força o modelo a aprender características invariantes das classes em vez de memorizar exemplos específicos.
- No nosso caso utilizamos dois tipos de data augmentation, o fornecido na diciplina e uma forma de data augmentation utilizando o ImageDataGenerator que tem um mais de parâmetros de pré processamento e de data augmentation

```python
# Configuração do ImageDataGenerator
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.2),
        layers.RandomZoom(0.2),
        layers.RandomContrast(0.2),
    ]
)

datagen = ImageDataGenerator(rescale=1./255)

IMG_SIZE = 32
BATCH_SIZE = 32
num_classes = 10

train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
```

```
    class_mode='categorical'
)

validation_dataset = datagen.flow_from_directory(
    validation_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
)

test_dataset = datagen.flow_from_directory(
    test_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
)

Found 40000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.
```

Arquitetura

- A data agumentation é uma função ou camada que aplica aumentação de dados, essencial para melhorar a generalização e robustez do modelo ao introduzir variações nos dados de treino.

Regularização L2

- Aplicamos nas camadas convolucionais e densas para penalizar os pesos grandes na função de *loss* durante o treino, o que ajuda a evitar *overfitting* ao reduzir a complexidade do modelo

Otimizador

- O modelo é compilado com o otimizador Adam (optimizer=Adam(learning_rate=0.0001)), que é amplamente utilizado devido à sua eficiência em ajustar as taxas de aprendizagem de forma adaptativa para cada parâmetro da rede neuronal e foi também o otimizador que mais utilizamos nada aulas

Função Loss

- A função de *loss* escolhida foi a categorical_crossentropy, adequada para problemas de classificação multiclasse

Métrica da avaliação

- A métrica de avaliação durante o treino foi a acurácia (metrics=['accuracy']), que mede a proporção de *predicts* corretas em relação ao total de previsões

Layers Convolucionis

- Cada uma das camadas convolucionais (Conv2D) é seguida pela ativação ReLU (activation='relu'), utilizando padding do tipo 'same' para manter o tamanho da saída igual ao da entrada (padding='same'). - Além disso, aplicámos regularização do kernel através do kernel_regularizer=l2(0.0001), que aplica regularização L2 para ajudar a evitar overfitting.
- Após cada camada convolucional, aplicámos normalização de batch (BatchNormalization) para acelerar o treino e melhorar a estabilidade do modelo.
- Posteriormente, é utilizado um pooling máximo (MaxPooling2D) com uma janela de (2, 2) para reduzir a dimensionalidade dos dados e extrair características mais importantes da imagem.

Camada Flatten

- Transforma a saída das camadas convolucionais em um vetor unidimensional para ligar a parte convolucional à *fully connected* da rede

Camadas Fully Connected

- A primeira camada densa (Dense) com 512 unidades, ativação ReLU, e regularização do kernel através do kernel_regularizer=l2(0.001). Normalização de batch (BatchNormalization) e dropout de 30% (Dropout(0.3)) para a regularização e prevenção do *overfitting*.

```
from tensorflow.keras.layers import GlobalAveragePooling2D
# Definindo o input
inputs = Input(shape=(IMG_SIZE, IMG_SIZE, 3))

# Aplicando Data Augmentation
x = data_augmentation(inputs)

# Primeira camada convolucional
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(0.0001))(inputs)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

# Segunda camada convolucional
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

# Terceira camada convolucional
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)
```

```
# Quarta camada convolucional
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.4)(x)

# Camada de Flatten
x = layers.Flatten()(x)

# Camada totalmente conectada
x = layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001))
(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
# Camada de saída
outputs = layers.Dense(10, activation='softmax')(x)  # Supondo 10
classes

# Definindo o modelo
model = Model(inputs=inputs, outputs=outputs)

# Compilando o modelo
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

model.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_16 (Conv2D)          (None, 32, 32, 64)        1792

 batch_normalization_20 (Ba  (None, 32, 32, 64)        256
 tchNormalization)

 max_pooling2d_16 (MaxPooli  (None, 16, 16, 64)        0
 ng2D)

 dropout_20 (Dropout)        (None, 16, 16, 64)        0

 conv2d_17 (Conv2D)          (None, 16, 16, 128)       73856

 batch_normalization_21 (Ba  (None, 16, 16, 128)       512
 tchNormalization)
```

```
 max_pooling2d_17 (MaxPooli  (None, 8, 8, 128)         0
 ng2D)

 dropout_21 (Dropout)        (None, 8, 8, 128)         0

 conv2d_18 (Conv2D)          (None, 8, 8, 256)         295168

 batch_normalization_22 (Ba  (None, 8, 8, 256)         1024
 tchNormalization)

 max_pooling2d_18 (MaxPooli  (None, 4, 4, 256)         0
 ng2D)

 dropout_22 (Dropout)        (None, 4, 4, 256)         0

 conv2d_19 (Conv2D)          (None, 4, 4, 512)         1180160

 batch_normalization_23 (Ba  (None, 4, 4, 512)         2048
 tchNormalization)

 max_pooling2d_19 (MaxPooli  (None, 2, 2, 512)         0
 ng2D)

 dropout_23 (Dropout)        (None, 2, 2, 512)         0

 flatten_4 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 512)               1049088

 batch_normalization_24 (Ba  (None, 512)               2048
 tchNormalization)

 dropout_24 (Dropout)        (None, 512)               0

 dense_9 (Dense)             (None, 10)                5130

=================================================================
Total params: 2611082 (9.96 MB)
Trainable params: 2608138 (9.95 MB)
Non-trainable params: 2944 (11.50 KB)
_________________________________________________________________
```

Callbacks

- É uma mais valia para os treinos foi o early_stopping que, se o val_loss nao mudar durante 10 epocas ele parar de treinar e fica com os melhores pesos que teve
- ReduceLr vai reduzindo o lr se mantiver o val_loss com 0.1 durante 5 vezes

```
# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
```

```
restore_best_weights=True)#Se o val_loss nao mudar durante 10 epocas
ele para de treianr e fica com os melhores pesos que teve
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=5, min_lr=0.000001)#Vai reduzindo o lr se mantiver o val_loss
com 0.1 durante 5 vezes

# Treinar o modelo
history = model.fit(train_generator,
epochs=100 ,validation_data=validation_dataset,
callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/100
1250/1250 [==============================] - 91s 72ms/step - loss:
3.2582 - accuracy: 0.2993 - val_loss: 2.6926 - val_accuracy: 0.4049 -
lr: 1.0000e-04
Epoch 2/100
1250/1250 [==============================] - 91s 72ms/step - loss:
2.6692 - accuracy: 0.4018 - val_loss: 2.5478 - val_accuracy: 0.4512 -
lr: 1.0000e-04
Epoch 3/100
1250/1250 [==============================] - 87s 70ms/step - loss:
2.3859 - accuracy: 0.4638 - val_loss: 2.4864 - val_accuracy: 0.4559 -
lr: 1.0000e-04
Epoch 4/100
1250/1250 [==============================] - 86s 69ms/step - loss:
2.1586 - accuracy: 0.5124 - val_loss: 2.1695 - val_accuracy: 0.5325 -
lr: 1.0000e-04
Epoch 5/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.9696 - accuracy: 0.5560 - val_loss: 1.8008 - val_accuracy: 0.6115 -
lr: 1.0000e-04
Epoch 6/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.8154 - accuracy: 0.5911 - val_loss: 1.6581 - val_accuracy: 0.6351 -
lr: 1.0000e-04
Epoch 7/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.6646 - accuracy: 0.6255 - val_loss: 1.5218 - val_accuracy: 0.6669 -
lr: 1.0000e-04
Epoch 8/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.5346 - accuracy: 0.6498 - val_loss: 1.8255 - val_accuracy: 0.5776 -
lr: 1.0000e-04
Epoch 9/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.4222 - accuracy: 0.6735 - val_loss: 1.5818 - val_accuracy: 0.6381 -
lr: 1.0000e-04
Epoch 10/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.3210 - accuracy: 0.6936 - val_loss: 1.2506 - val_accuracy: 0.7137 -
```

```
lr: 1.0000e-04
Epoch 11/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.2356 - accuracy: 0.7098 - val_loss: 1.2415 - val_accuracy: 0.7111 -
lr: 1.0000e-04
Epoch 12/100
1250/1250 [==============================] - 86s 68ms/step - loss:
1.1684 - accuracy: 0.7227 - val_loss: 1.0265 - val_accuracy: 0.7672 -
lr: 1.0000e-04
Epoch 13/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.0926 - accuracy: 0.7386 - val_loss: 1.0178 - val_accuracy: 0.7650 -
lr: 1.0000e-04
Epoch 14/100
1250/1250 [==============================] - 86s 69ms/step - loss:
1.0366 - accuracy: 0.7478 - val_loss: 0.9121 - val_accuracy: 0.7921 -
lr: 1.0000e-04
Epoch 15/100
1250/1250 [==============================] - 87s 69ms/step - loss:
0.9810 - accuracy: 0.7608 - val_loss: 0.9076 - val_accuracy: 0.7861 -
lr: 1.0000e-04
Epoch 16/100
1250/1250 [==============================] - 89s 71ms/step - loss:
0.9452 - accuracy: 0.7678 - val_loss: 0.8903 - val_accuracy: 0.7872 -
lr: 1.0000e-04
Epoch 17/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.9024 - accuracy: 0.7782 - val_loss: 0.9245 - val_accuracy: 0.7749 -
lr: 1.0000e-04
Epoch 18/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.8621 - accuracy: 0.7891 - val_loss: 1.0631 - val_accuracy: 0.7286 -
lr: 1.0000e-04
Epoch 19/100
1250/1250 [==============================] - 94s 75ms/step - loss:
0.8315 - accuracy: 0.7947 - val_loss: 1.0109 - val_accuracy: 0.7502 -
lr: 1.0000e-04
Epoch 20/100
1250/1250 [==============================] - 98s 78ms/step - loss:
0.7952 - accuracy: 0.8051 - val_loss: 0.8664 - val_accuracy: 0.7887 -
lr: 1.0000e-04
Epoch 21/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.7713 - accuracy: 0.8132 - val_loss: 0.8028 - val_accuracy: 0.8047 -
lr: 1.0000e-04
Epoch 22/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.7488 - accuracy: 0.8170 - val_loss: 0.8505 - val_accuracy: 0.7903 -
lr: 1.0000e-04
```

```
Epoch 23/100
1250/1250 [==============================] - 88s 70ms/step - loss:
0.7263 - accuracy: 0.8240 - val_loss: 0.7608 - val_accuracy: 0.8174 -
lr: 1.0000e-04
Epoch 24/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.7083 - accuracy: 0.8306 - val_loss: 0.7760 - val_accuracy: 0.8093 -
lr: 1.0000e-04
Epoch 25/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.6833 - accuracy: 0.8375 - val_loss: 0.7990 - val_accuracy: 0.8024 -
lr: 1.0000e-04
Epoch 26/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.6705 - accuracy: 0.8403 - val_loss: 0.7650 - val_accuracy: 0.8126 -
lr: 1.0000e-04
Epoch 27/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.6538 - accuracy: 0.8458 - val_loss: 0.8486 - val_accuracy: 0.7871 -
lr: 1.0000e-04
Epoch 28/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.6394 - accuracy: 0.8517 - val_loss: 0.7769 - val_accuracy: 0.8059 -
lr: 1.0000e-04
Epoch 29/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5836 - accuracy: 0.8706 - val_loss: 0.7259 - val_accuracy: 0.8241 -
lr: 2.0000e-05
Epoch 30/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5688 - accuracy: 0.8734 - val_loss: 0.7165 - val_accuracy: 0.8278 -
lr: 2.0000e-05
Epoch 31/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5615 - accuracy: 0.8768 - val_loss: 0.7272 - val_accuracy: 0.8257 -
lr: 2.0000e-05
Epoch 32/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5415 - accuracy: 0.8812 - val_loss: 0.6982 - val_accuracy: 0.8356 -
lr: 2.0000e-05
Epoch 33/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5338 - accuracy: 0.8839 - val_loss: 0.6965 - val_accuracy: 0.8338 -
lr: 2.0000e-05
Epoch 34/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5196 - accuracy: 0.8877 - val_loss: 0.7057 - val_accuracy: 0.8302 -
lr: 2.0000e-05
Epoch 35/100
```

```
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5199 - accuracy: 0.8865 - val_loss: 0.7008 - val_accuracy: 0.8330 -
lr: 2.0000e-05
Epoch 36/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.5068 - accuracy: 0.8918 - val_loss: 0.7060 - val_accuracy: 0.8302 -
lr: 2.0000e-05
Epoch 37/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4970 - accuracy: 0.8934 - val_loss: 0.6934 - val_accuracy: 0.8341 -
lr: 2.0000e-05
Epoch 38/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4943 - accuracy: 0.8935 - val_loss: 0.6848 - val_accuracy: 0.8355 -
lr: 2.0000e-05
Epoch 39/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4849 - accuracy: 0.8960 - val_loss: 0.6785 - val_accuracy: 0.8393 -
lr: 2.0000e-05
Epoch 40/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4790 - accuracy: 0.8973 - val_loss: 0.6824 - val_accuracy: 0.8375 -
lr: 2.0000e-05
Epoch 41/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4775 - accuracy: 0.8973 - val_loss: 0.6955 - val_accuracy: 0.8338 -
lr: 2.0000e-05
Epoch 42/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4622 - accuracy: 0.9014 - val_loss: 0.6924 - val_accuracy: 0.8352 -
lr: 2.0000e-05
Epoch 43/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4596 - accuracy: 0.9033 - val_loss: 0.6948 - val_accuracy: 0.8320 -
lr: 2.0000e-05
Epoch 44/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4561 - accuracy: 0.9021 - val_loss: 0.6905 - val_accuracy: 0.8346 -
lr: 2.0000e-05
Epoch 45/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4438 - accuracy: 0.9077 - val_loss: 0.6852 - val_accuracy: 0.8366 -
lr: 4.0000e-06
Epoch 46/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4465 - accuracy: 0.9056 - val_loss: 0.6754 - val_accuracy: 0.8394 -
lr: 4.0000e-06
Epoch 47/100
1250/1250 [==============================] - 86s 69ms/step - loss:
```

```
0.4426 - accuracy: 0.9080 - val_loss: 0.6760 - val_accuracy: 0.8393 -
lr: 4.0000e-06
Epoch 48/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4362 - accuracy: 0.9088 - val_loss: 0.6791 - val_accuracy: 0.8388 -
lr: 4.0000e-06
Epoch 49/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4356 - accuracy: 0.9095 - val_loss: 0.6676 - val_accuracy: 0.8427 -
lr: 4.0000e-06
Epoch 50/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4323 - accuracy: 0.9106 - val_loss: 0.6776 - val_accuracy: 0.8389 -
lr: 4.0000e-06
Epoch 51/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4363 - accuracy: 0.9095 - val_loss: 0.6728 - val_accuracy: 0.8395 -
lr: 4.0000e-06
Epoch 52/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4390 - accuracy: 0.9065 - val_loss: 0.6741 - val_accuracy: 0.8396 -
lr: 4.0000e-06
Epoch 53/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4275 - accuracy: 0.9109 - val_loss: 0.6763 - val_accuracy: 0.8401 -
lr: 4.0000e-06
Epoch 54/100
1250/1250 [==============================] - 89s 71ms/step - loss:
0.4272 - accuracy: 0.9104 - val_loss: 0.6707 - val_accuracy: 0.8409 -
lr: 4.0000e-06
Epoch 55/100
1250/1250 [==============================] - 88s 71ms/step - loss:
0.4283 - accuracy: 0.9115 - val_loss: 0.6719 - val_accuracy: 0.8401 -
lr: 1.0000e-06
Epoch 56/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4241 - accuracy: 0.9134 - val_loss: 0.6708 - val_accuracy: 0.8416 -
lr: 1.0000e-06
Epoch 57/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4237 - accuracy: 0.9121 - val_loss: 0.6717 - val_accuracy: 0.8409 -
lr: 1.0000e-06
Epoch 58/100
1250/1250 [==============================] - 89s 72ms/step - loss:
0.4284 - accuracy: 0.9122 - val_loss: 0.6731 - val_accuracy: 0.8409 -
lr: 1.0000e-06
Epoch 59/100
1250/1250 [==============================] - 86s 69ms/step - loss:
0.4280 - accuracy: 0.9114 - val_loss: 0.6732 - val_accuracy: 0.8409 -
```

```
lr: 1.0000e-06
Epoch 60/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4240 - accuracy: 0.9133 - val_loss: 0.6730 - val_accuracy: 0.8404 -
lr: 1.0000e-06
Epoch 61/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4244 - accuracy: 0.9115 - val_loss: 0.6701 - val_accuracy: 0.8398 -
lr: 1.0000e-06
Epoch 62/100
1250/1250 [==============================] - 87s 69ms/step - loss:
0.4179 - accuracy: 0.9140 - val_loss: 0.6729 - val_accuracy: 0.8400 -
lr: 1.0000e-06
Epoch 63/100
1250/1250 [==============================] - 87s 69ms/step - loss:
0.4268 - accuracy: 0.9113 - val_loss: 0.6721 - val_accuracy: 0.8402 -
lr: 1.0000e-06
Epoch 64/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4206 - accuracy: 0.9140 - val_loss: 0.6703 - val_accuracy: 0.8411 -
lr: 1.0000e-06
Epoch 65/100
1250/1250 [==============================] - 89s 72ms/step - loss:
0.4206 - accuracy: 0.9156 - val_loss: 0.6697 - val_accuracy: 0.8417 -
lr: 1.0000e-06
Epoch 66/100
1250/1250 [==============================] - 88s 70ms/step - loss:
0.4135 - accuracy: 0.9167 - val_loss: 0.6703 - val_accuracy: 0.8405 -
lr: 1.0000e-06
Epoch 67/100
1250/1250 [==============================] - 89s 71ms/step - loss:
0.4233 - accuracy: 0.9125 - val_loss: 0.6711 - val_accuracy: 0.8409 -
lr: 1.0000e-06
Epoch 68/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4181 - accuracy: 0.9134 - val_loss: 0.6701 - val_accuracy: 0.8400 -
lr: 1.0000e-06
Epoch 69/100
1250/1250 [==============================] - 87s 70ms/step - loss:
0.4177 - accuracy: 0.9137 - val_loss: 0.6711 - val_accuracy: 0.8410 -
lr: 1.0000e-06

#Saving the model
model.save('From_Scratch_Com_DataAugmentation.h5')

from tensorflow import keras

model =
keras.models.load_model('From_Scratch_Com_DataAugmentation.h5')
```

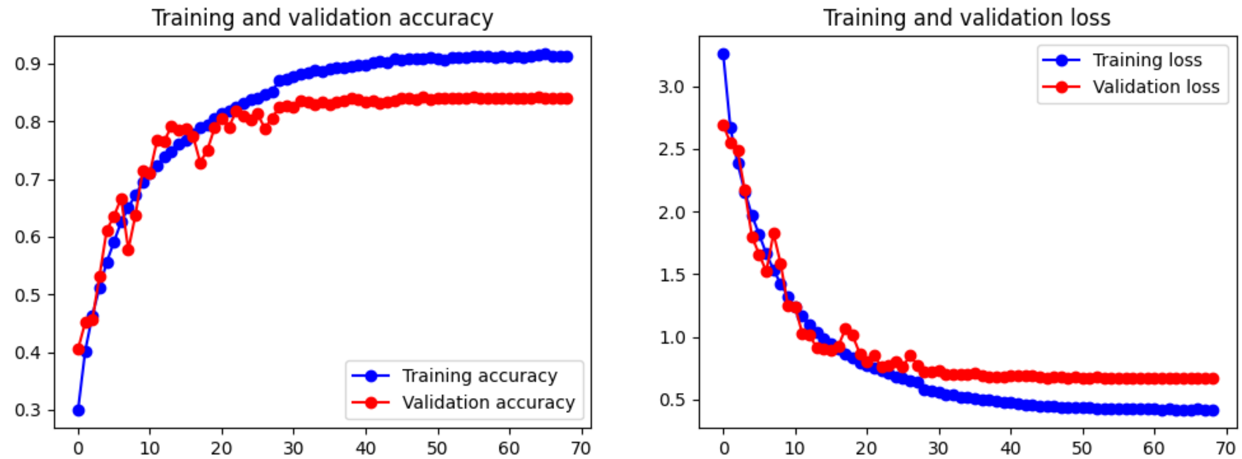
```
# Validacao da Rede
val_loss, val_acc = model.evaluate(validation_dataset)
print('val_acc:', val_acc)

# Avaliar o modelo
test_loss, test_acc = model.evaluate(test_dataset)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 6s 20ms/step - loss: 0.6980
- accuracy: 0.8531
val_acc: 0.8531000018119812
313/313 [==============================] - 6s 20ms/step - loss: 0.7364
- accuracy: 0.8455
Test accuracy: 0.8454999923706055
```

Análise dos gráficos

- Os gráficos resultantes do treino demostram que a rede está praticamente com *overfitting* nulo o que é bom, embora o desempenho da mesma não seja o esperado

```
# Plotando os resultados
import matplotlib.pyplot as plt

def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'bo-', label='Training accuracy')
    plt.plot(epochs, val_acc, 'ro-', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'bo-', label='Training loss')
    plt.plot(epochs, val_loss, 'ro-', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

plot_training_history(history)
```



Training and validation accuracy
Training accuracy
Validation accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
50
60
70
Training and validation loss
Training loss
Validation loss
0.5
1.0
1.5
2.0
2.5
3.0
0
10
20
30
40
50
60
70