

Lab 6 - BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Convolução e CNN

Prof. Eduardo e Prof. Pedro

Objetivos:

- Aplicação de filtros em imagens por meio de convolução
- Entendimento do uso de stride, padding e pooling
- Modelagem de uma rede de convolução para o problema de rec. de face da AT&T
- Uso do VGG pr-e-treinado como um extrator de características
- Uso do MobileNet pré-treinado para classificação de faces : transferência de aprendizagem
- Notebook baseado em tensorflow e Keras.

Data da entrega : 21/11

- Complete o código (marcado com ToDo) e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-LabX.pdf"
- Envie o PDF via google [FORM](#)

▼ 1. Aplicando filtros e entendendo padding, stride e pooling (20pt)

▼ 1.1. Importando pacotes e montando o drive

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, AvgPool2D
from tensorflow.keras import datasets, layers, models
import os
import skimage
from skimage import io
import numpy as np

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

from skimage.io import imread
from skimage.transform import resize
```

▼ 1.2. Carregando uma imagem

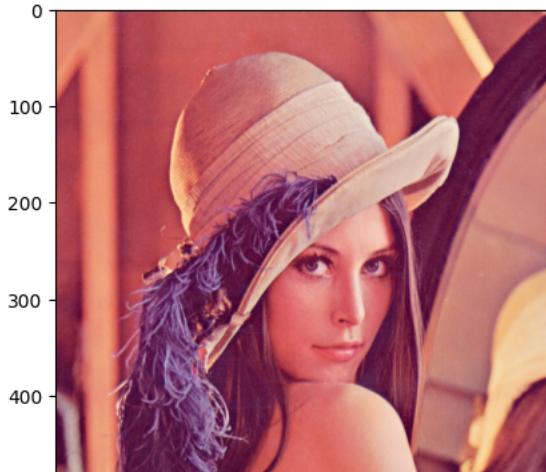
Carregue um imagem do disco, para usar como exemplo.

```
# carrega imagem de exemplo
sample_image = imread("/content/drive/My Drive/Lenna.png")
sample_image= sample_image.astype(float)

size = sample_image.shape
print("sample image shape: ", sample_image.shape)

plt.imshow(sample_image.astype('uint8'));
```

```
sample image shape: (512, 512, 3)
```



```
# veja o shape da imagem
sample_image.shape
```

```
(512, 512, 3)
```

▼ 1.3. Criando e aplicando um filtro com convolução

Utilize o tf/Keras para aplicar o filtro. Observe que nesta etapa não há necessidade de treinamento algum. O código abaixo cria 3 filtros de tamanho 5x5, e adiciona padding de forma a manter a imagem de saída (filtrada) do mesmo tamanho da imagem de entrada (padding = "same").

```
#cria um objeto sequencial com apenas uma camada de convolução do tipo tf.keras.layers.Conv2D
conv = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3))
])
conv.output_shape
```

```
(None, None, None, 3)
```

```
conv.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, None, None, 3)	228
<hr/>		
Total params: 228 (912.00 Byte)		
Trainable params: 228 (912.00 Byte)		
Non-trainable params: 0 (0.00 Byte)		

```
# com TF/kertas, as convoluções esperam vetores no formato : (batch_size, dim1, dim2, dim3). Ou seja, a primeira posição é o tamanho do
# Uma imagem isolada é considerada um lote de tamanho 1, portanto, deve-se expandir mais uma dimensão do tensor.
img_in = np.expand_dims(sample_image, 0)
img_in.shape
```

```
(1, 512, 512, 3)
```

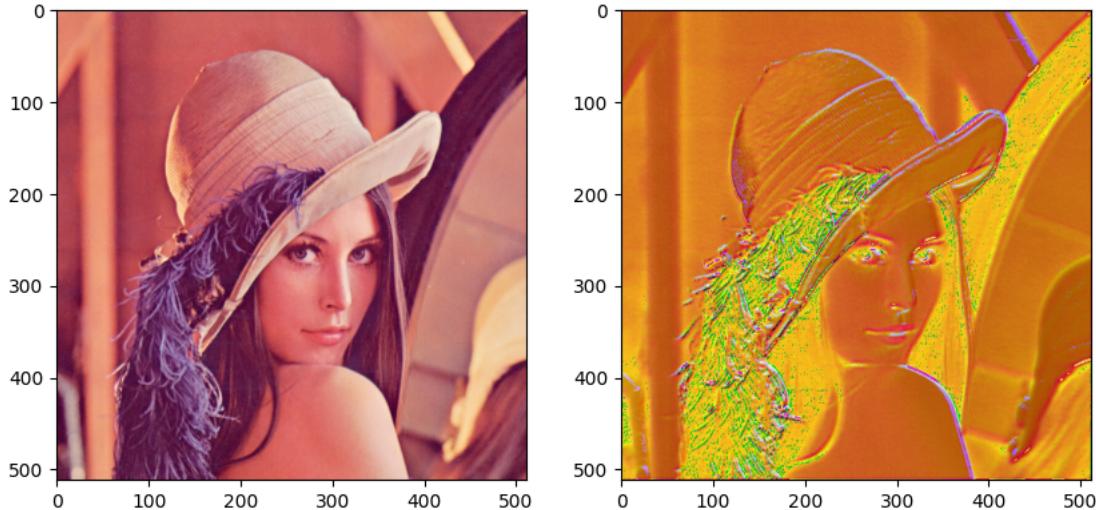
Agora, pode-se aplicar a convolução. Aplique a convolução na imagem de exemplo (expandida) e verifique o tamanho da imagem resultante (img_out). Use a função predict do objeto conv para aplicar a convolução.

```
img_out = conv(img_in)
img_out.shape
```

```
TensorShape([1, 512, 512, 3])
```

Plote as imagens lado a lado e observe o resultado. O parâmetro "same" no padding aplica um padding automático no sentido de garantir que a saída tenha o mesmo tamanho da entrada. Lembre-se que o padding adiciona zeros nas bordas da imagem, antes da aplicação da convolução.

```
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))
ax0.imshow(sample_image.astype('uint8'))
ax1.imshow(img_out[0].numpy().astype('uint8'));
```



Repita o mesmo procedimento, trocando padding de 'same' para 'valid', usando apenas um filtro.

```
conv2 = Sequential([
    Conv2D(filters=1, kernel_size=(5, 5), padding="valid",
           input_shape=(None, None, 3))
])
conv2.output_shape

(None, None, None, 1)
```

```
conv2.summary() # 1 filtro 5x5x3 ... a profundidade do filtro é de acordo com a entrada. 5x5x3 = 75; Não esqueça do bias!
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, None, None, 1)	76
<hr/>		
Total params: 76 (304.00 Byte)		
Trainable params: 76 (304.00 Byte)		
Non-trainable params: 0 (0.00 Byte)		

```
img_out = conv2(img_in)
img_out[0].shape
```

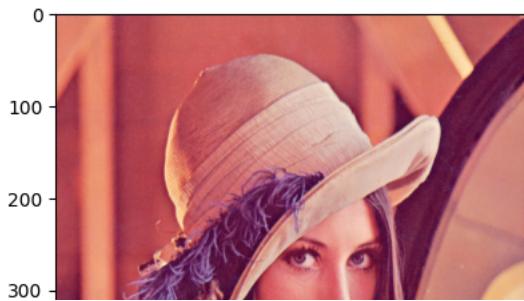
```
TensorShape([508, 508, 1])
```

Plote as duas imagens lado a lado

```
# Como tivemos que expandir a primeira dimensão para aplicar a convolução, podemos remover a dimensão unitária para plotar a imagem, u
i = img_out[0].numpy().squeeze()
i.shape

(508, 508)
```

```
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))
ax0.imshow(sample_image.astype('uint8'))
i = img_out[0].numpy().squeeze()
ax1.imshow(i.astype('uint8'));
```



▼ 1.4. Inicializando os filtros na mão

```
0 100 200 300 400 500 0 100 200 300 400 500
```

A função abaixo inicializa um array de dimensões 5,5,3,3 com todas as posições zero, exceto as posições 5,5,0,0 , 5,5,1,1 e 5,5,2,2 que recebem o valor 1/25.

```
def my_filter(shape=(5, 5, 3, 3), dtype=None):
    array = np.zeros(shape=shape, dtype=np.float32)
    array[:, :, 0, 0] = 1 / 25
    array[:, :, 1, 1] = 1 / 25
    array[:, :, 2, 2] = 1 / 25
    return array

# transposição apenas para ajudar na visualização
np.transpose(my_filter(), (2, 3, 0, 1))

array([[[[0.04, 0.04, 0.04, 0.04, 0.04],
          [0.04, 0.04, 0.04, 0.04, 0.04],
          [0.04, 0.04, 0.04, 0.04, 0.04],
          [0.04, 0.04, 0.04, 0.04, 0.04],
          [0.04, 0.04, 0.04, 0.04, 0.04]],

         [[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]],

         [[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]],

         [[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]],

         [[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]],

         [[[0.04, 0.04, 0.04, 0.04, 0.04],
           [0.04, 0.04, 0.04, 0.04, 0.04],
           [0.04, 0.04, 0.04, 0.04, 0.04],
           [0.04, 0.04, 0.04, 0.04, 0.04],
           [0.04, 0.04, 0.04, 0.04, 0.04]],

          [[0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.]],

          [[0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.]],

          [[0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.]],

          [[0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0.]],

          [[[0.04, 0.04, 0.04, 0.04, 0.04],
            [0.04, 0.04, 0.04, 0.04, 0.04],
            [0.04, 0.04, 0.04, 0.04, 0.04],
            [0.04, 0.04, 0.04, 0.04, 0.04],
            [0.04, 0.04, 0.04, 0.04, 0.04]],

           [[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]],

           [[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]],

           [[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]],

           [[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]]], dtype=float32)
```

```
# a função definida acima é usada para carregar valores nos filtros.
# use a função my_filter() para pre-inicializar os filtros do objeto conv3.
#
conv3 = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3), kernel_initializer=my_filter)
])
conv3.output_shape

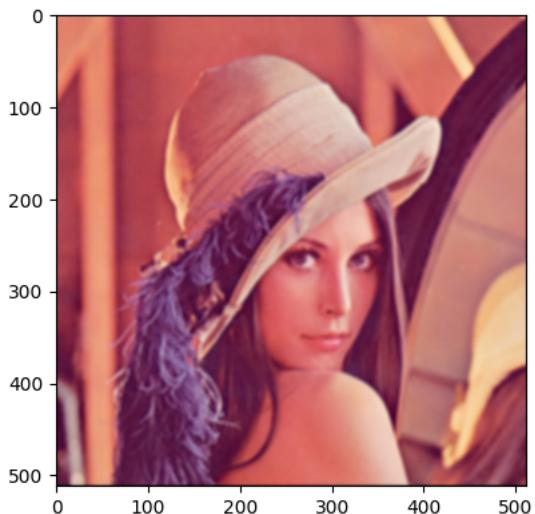
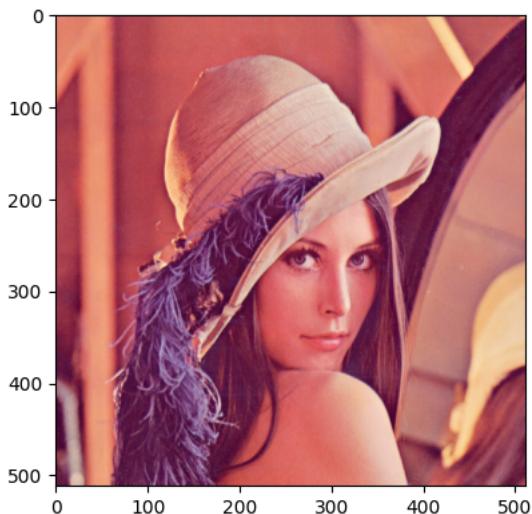
(None, None, None, 3)
```

▼ 1.5. Plote e observe o que acontece com a imagem (1pt)

Foi observada uma redução da nitidez na imagem.

```
# observe o que acontece com a imagem
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))
ax0.imshow(img_in[0].astype('uint8'))
ax1.imshow(conv3.predict(img_in)[0].astype('uint8'));
```

1/1 [=====] - 0s 98ms/step



Responda

ToDo : Descreva suas observações sobre a imagem anterior.

▼ 1.6. Filtros de borda (5pt)

ToDo : Crie uma nova função para gerar um filtro de borda nos 3 canais da imagem de entrada. O filtro deve ser 3x3 e ter o formato [[0 0.2 0] [0 -0.2 0] [0 0 0]] (2pt)

```
def my_new_filter(shape=(1, 3, 3, 3), dtype=None):
    array = np.zeros(shape=shape, dtype=np.float32)
    array[:, 1, :, 0] = 1 / 5
    array[:, 1, :, 1] = -1 / 5
    return array

np.transpose(my_new_filter(), (2, 3, 0, 1))

array([[[[ 0.,  0.2,  0. ]],
       [[ 0., -0.2,  0. ]],
       [[ 0.,  0.,  0. ]]],
      [[[ 0.,  0.2,  0. ]],
       [[ 0., -0.2,  0. ]],
       [[ 0.,  0.,  0. ]]]],
```

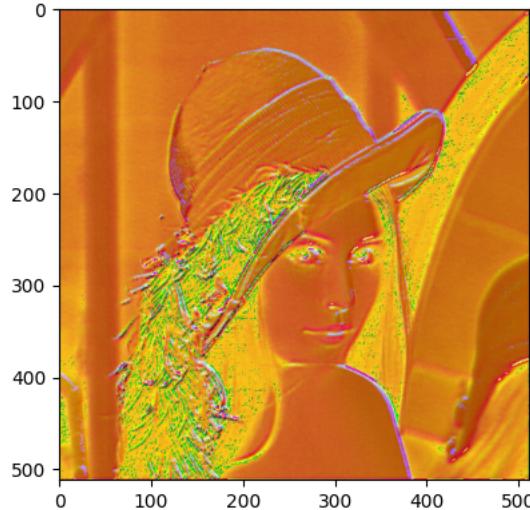
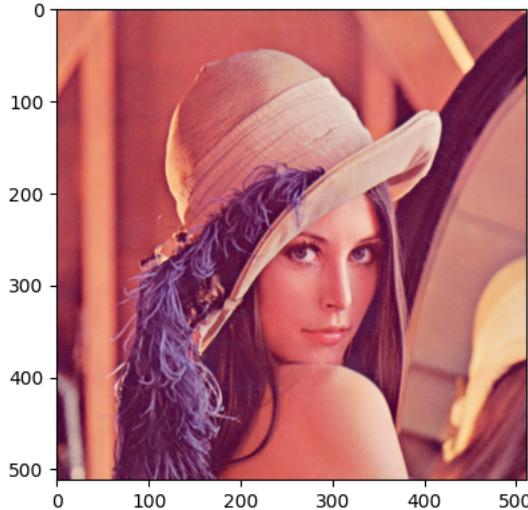
```
[[[ 0. ,  0.2,  0. ]],  
 [[ 0. , -0.2,  0. ]],  
 [[ 0. ,  0. ,  0. ]]], dtype=float32)
```

Inicialize o objeto conv4 com seu novo filtro e aplique na imagem de entrada

```
conv4 = Sequential(  
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",  
           input_shape=(None, None, 3), kernel_initializer=my_new_filter)  
)  
conv4.output_shape  
  
(None, None, None, 3)
```

```
# Plote as duas iamgens lado a lado (filtrada e não filtrada)  
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))  
ax0.imshow(img_in[0].astype('uint8'))  
ax1.imshow(conv4.predict(img_in)[0].astype('uint8'));
```

1/1 [=====] - 0s 55ms/step



▼ 1.7. Pooling (14pt)

Aplique um max-pooling na imagem, com uma janela de 2x2. Faça com stride de 2 e observe o resultado na imagem de saída.

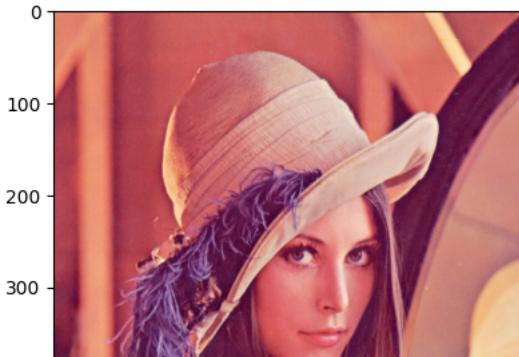
```
# cria um objeto Sequencial de nome max_pool, apenas contendo uma camada de tf.keras.layers.MaxPool2D.  
# lembre-se de colocar o parametro input_shape como input_shape=(None, None, 3)
```

```
max_pool = Sequential(  
    MaxPool2D(pool_size=(2, 2), strides=2, input_shape=(None, None, 3))  
)  
max_pool.output_shape  
  
(None, None, None, 3)
```

```
img_in = np.expand_dims(sample_image, 0) # expande a imagem  
img_out = max_pool.predict(img_in) # aplica o pooling
```

1/1 [=====] - 0s 51ms/step

```
# plota as imagens lado a lado  
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))  
ax0.imshow(img_in[0].astype('uint8'))  
ax1.imshow(img_out[0].astype('uint8'));
```



Aumente o stride para 4, repita o processo e observe o resultado na imagem de saída.



```
# cria um objeto Sequencial de nome max_pool, apenas contendo uma camada de tf.keras.layer.MaxPool2D.
# lembre-se de colocar o parametro input-shape como input_shape=(None, None, 3)
# Coloque o parametro stride para 4
```

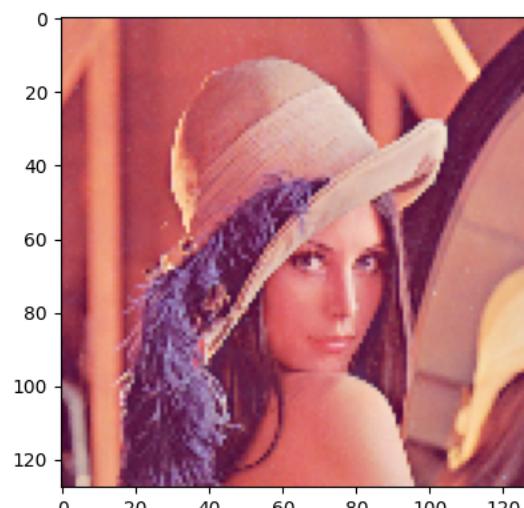
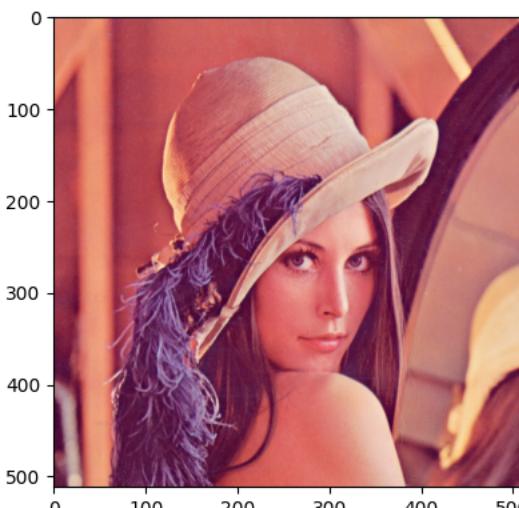
```
max_pool2 = Sequential(
    MaxPool2D(pool_size=(2, 2), strides=4, input_shape=(None, None, 3))
)
max_pool2.output_shape
```

```
(None, None, None, 3)

img_in = np.expand_dims(sample_image, 0) # expande a imagem
img_out = max_pool2.predict(img_in) # aplica o pooling
```

```
1/1 [=====] - 0s 62ms/step
```

```
# plota as imagens lado a lado
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))
ax0.imshow(img_in[0].astype('uint8'))
ax1.imshow(img_out[0].astype('uint8'));
```



Aumente o stride para 8, repita o processo e observe o resultado na imagem de saída. A

```
# cria um objeto Sequencial de nome max_pool, apenas contendo uma camada de tf.keras.layer.MaxPool2D.
# lembre-se de colocar o parametro input-shape como input_shape=(None, None, 3)
# Coloque o parametro stride para 4
```

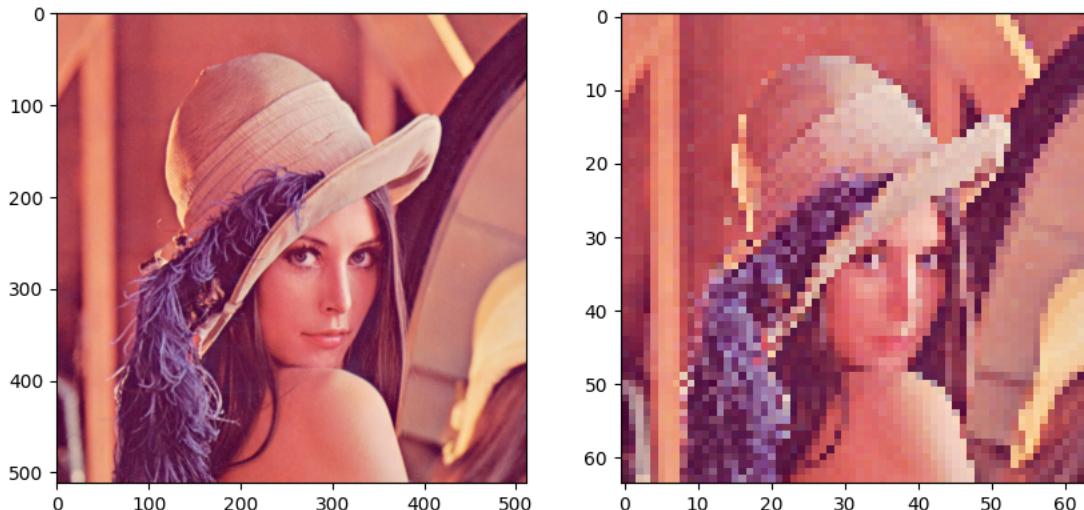
```
max_pool3 = Sequential(
    MaxPool2D(pool_size=(2, 2), strides=8, input_shape=(None, None, 3))
)
max_pool3.output_shape
```

```
(None, None, None, 3)

img_in = np.expand_dims(sample_image, 0) # expande a imagem
img_out = max_pool3.predict(img_in) # aplica o pooling
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7cc15f4f6200>
1/1 [=====] - 0s 56ms/step
```

```
# plota as imagens lado a lado
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10, 5))
ax0.imshow(img_in[0].astype('uint8'))
ax1.imshow(img_out[0].astype('uint8'));
```



Responda

ToDo - Descreva o que aconteceu com o aumento do stride.

▼ 2. Reconhecimento de Faces usando uma rede de convolução (20pt)

O objetivo desta etapa é classificar faces na base ORL (AT&T) Database (40 individuos x 10 imagens, de resolução 92x112 pixels e 256 níveis de cinza).

Baixe as imagens no site <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html> ou da pasta dataset do Drive.

▼ 2.1. Preparando os dados (5pt)

```
# inicializa matrizes X e y
X = np.empty([400, 112, 92]) # 40 classe com 10 imgs cada, 10304 = 112x92
y = np.empty([400, 1])

# percorre todos os diretorios da base att e carrega as imagens
imgs_path = "/content/drive/MyDrive/AttFaces"
i=0
class_id = 0
for f in os.listdir(imgs_path):
    #print(f)
    if f.startswith("s"):
        class_id = class_id + 1
        for img_path in os.listdir(os.path.join(imgs_path,f)):
            if img_path.endswith(".pgm"):
                #print(img_path)
                X[i, :, :] = io.imread(os.path.join(imgs_path,f,img_path))
                y[i, :] = class_id
                i = i + 1

print("dimensões da matriz X = " , X.shape)

dimensões da matriz X = (400, 112, 92)

# Divila os dados em treino e teste (70%-30%) com a função train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

X_train.shape

(280, 112, 92)
```

```
X_test.shape
```

```
(120, 112, 92)
```

▼ 2.2. Implementando a rede (15pt)

Implemente uma rede de convolução simples, contendo 3 camadas de convolução seguidas de camadas max-pooling. Duas camadas densas (totalmente conectadas) no final e por fim uma camada com ativação softmax para a classificação. Escolha filtros de tamanhos variados: (3,3) ou (5,5). Para cada camada, crie de 32 a 96 filtros. Na camada densa, use de 64 a 200 neurônios.

Use o comando `model.summary()` para conferir a arquitetura.

```
# Implementa uma rede de convolução simples, chamada model
input_size = (X.shape[1], X.shape[2], 1)
n_classes = 40

model = models.Sequential()

model.add(layers.InputLayer(input_shape=input_size))

model.add(layers.Conv2D(filters=32, kernel_size=(5, 5), padding="same", input_shape=(None, None, 3)))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, input_shape=(None, None, 3)))

model.add(layers.Conv2D(filters=96, kernel_size=(3, 3), padding="same", input_shape=(None, None, 3)))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, input_shape=(None, None, 3)))

model.add(layers.Conv2D(filters=96, kernel_size=(3, 3), padding="same", input_shape=(None, None, 3)))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=2, input_shape=(None, None, 3)))

model.add(layers.Flatten())

model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(80, activation='relu'))
model.add(layers.Dense(n_classes, activation='softmax'))

model.summary()

Model: "sequential_7"

Layer (type)          Output Shape         Param #
================================================================
conv2d_4 (Conv2D)     (None, 112, 92, 32)    832
max_pooling2d_3 (MaxPooling2D) (None, 56, 46, 32)    0
conv2d_5 (Conv2D)     (None, 56, 46, 96)    27744
max_pooling2d_4 (MaxPooling2D) (None, 28, 23, 96)    0
conv2d_6 (Conv2D)     (None, 28, 23, 96)    83040
max_pooling2d_5 (MaxPooling2D) (None, 14, 11, 96)    0
flatten (Flatten)    (None, 14784)        0
dense (Dense)        (None, 128)          1892480
dense_1 (Dense)      (None, 80)           10320
dense_2 (Dense)      (None, 40)           3240
=====
Total params: 2017656 (7.70 MB)
Trainable params: 2017656 (7.70 MB)
Non-trainable params: 0 (0.00 Byte)
```

Seu modelo deve ter uma saída aproximadamente como abaixo:

```
Model: "sequential_36"
```

Layer (type)	Output Shape	Param #
conv2d_60 (Conv2D)	(None, 110, 90, 32)	320

```

max_pooling2d_18 (MaxPooling (None, 55, 45, 32)          0
=====
conv2d_61 (Conv2D)           (None, 53, 43, 64)        18496
=====
max_pooling2d_19 (MaxPooling (None, 26, 21, 64)        0
=====
conv2d_62 (Conv2D)           (None, 24, 19, 64)        36928
=====
flatten_5 (Flatten)          (None, 29184)            0
=====
dense_9 (Dense)              (None, 64)                1867840
=====
dense_10 (Dense)             (None, 40)                2600
=====
Total params: 1,926,184
Trainable params: 1,926,184
Non-trainable params: 0
=====
```

```
# repare bem o shape de x_train. A priumeira dimensão é o tamamho do lote, a segunda e terceira são referentes ao taamnho das imagens.
# repare que as imagens desta base tem apenas uma banda (escala de cinza)
X_train.shape
```

```
(280, 112, 92)
```

```
# Como o tensor acima não contempla o tamamho de canais (no caso , igual a 1), deve-se expandir a última dimensão para deixar a entrada
X_train_new = np.expand_dims(X_train, axis=-1)
X_test_new = np.expand_dims(X_test, axis=-1)
```

```
X_train_new.shape
```

```
(280, 112, 92, 1)
```

```
# o vetor de rótulos não precisa ter duas diemnões.
y_train_new = y_train.squeeze()
y_test_new = y_test.squeeze()
```

```
# e deve ficar na faixa entre 0 e 39
y_train_new = y_train_new - 1;
y_test_new = y_test_new - 1;
```

Compile o modelo usando o método de otimização=adam e função de custo (loss) = sparse_categorical_crossentropy.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Treine o modelo por 30 épocas com batch_size = 100.

```
history = model.fit(X_train_new,
                     y_train_new,
                     epochs=30,
                     batch_size=100,
                     validation_data=(X_test_new, y_test_new))

3/3 [=====] - 0s 69ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 3/30
3/3 [=====] - 0s 53ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 4/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 5/30
3/3 [=====] - 0s 49ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 6/30
3/3 [=====] - 0s 82ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 7/30
3/3 [=====] - 0s 65ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 8/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 9/30
3/3 [=====] - 0s 49ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 10/30
```

```
Epoch 12/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 13/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 14/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 15/30
3/3 [=====] - 0s 52ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 16/30
3/3 [=====] - 0s 50ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 17/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 18/30
3/3 [=====] - 0s 50ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 19/30
3/3 [=====] - 0s 70ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 20/30
3/3 [=====] - 0s 50ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 21/30
3/3 [=====] - 0s 70ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 22/30
3/3 [=====] - 0s 49ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 23/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 24/30
3/3 [=====] - 0s 49ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 25/30
3/3 [=====] - 0s 51ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 26/30
3/3 [=====] - 0s 49ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 27/30
3/3 [=====] - 0s 52ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 28/30
3/3 [=====] - 0s 53ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 29/30
3/3 [=====] - 0s 50ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 30/30
3/3 [=====] - 0s 52ms/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
```

O retorno da função fit() é um objeto para armazenar o histórico do treino.

```
history.history.keys()
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Plote a acurácia e o custo (loss) do treino e da validação.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(X_test_new, y_test_new, verbose=2)
```

```
4/4 - 0s - loss: nan - accuracy: 0.0000e+00 - 351ms/epoch - 88ms/step
```



▼ 3. Usando um modelo Pré-treinado : VGG (10pt)

Carregando os dados da base AT&T para o VGG. Como a base está em escala de cinza e a entrada do modelo VGG espera uma imagem colorida (RGB), vamos repetir a mesma imagem em cada uma das bandas.

Step	val accuracy
0	0.6
1	1.0
2	1.0
3	1.0
4	1.0
5	1.0
6	1.0
7	1.0
8	1.0
9	1.0
10	1.0
11	1.0
12	1.0
13	1.0
14	1.0
15	1.0
16	1.0
17	1.0
18	1.0
19	1.0
20	1.0
21	1.0
22	1.0
23	1.0
24	1.0
25	1.0
26	1.0
27	1.0
28	1.0
29	1.0
30	1.0

▼ 3.1. Preparando os dados (2pt)

```
# inicializa matrizes X e y
X = np.empty([400, 112, 92, 3]) # 40 classe com 10 imgs cada, 10304 = 112x92
y = np.empty([400, 1])

# percorre todos os diretorios da base att e carrega as imagens
imgs_path = "/content/drive/MyDrive/AttFaces"
i=0
class_id = 0
for f in os.listdir(imgs_path):
    #print(f)
    if f.startswith("s"):
        class_id = class_id + 1
    for img_path in os.listdir(os.path.join(imgs_path,f)):
        if img_path.endswith(".pgm"):
            #print(img_path)
            # copia msg imagem para os 3 canais
            X[i, :, :,0] = io.imread(os.path.join(imgs_path,f,img_path))
            X[i, :, :,1] = io.imread(os.path.join(imgs_path,f,img_path))
            X[i, :, :,2] = io.imread(os.path.join(imgs_path,f,img_path))
            y[i, :] = class_id-1
            i = i + 1

# divida em 70% treino e 30% teste
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train.shape

(280, 112, 92, 3)
```

▼ 3.2. Carrando o VGG direto da biblioteca do tensorflow (2pt)

```
# https://www.tensorflow.org/guide/keras/functional?hl=pt_br

from tensorflow.keras.applications import VGG19
vgg19 = VGG19()

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19\_weights\_tf\_dim\_ordering\_tf\_kernels.h5
574710816/574710816 [=====] - 7s 0us/step
```

```
< >
vgg19.summary() # repare a quantidade de parâmetros!
```

```
=====
input_2 (InputLayer)      [(None, 224, 224, 3)]      0
block1_conv1 (Conv2D)     (None, 224, 224, 64)      1792
block1_conv2 (Conv2D)     (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D) (None, 112, 112, 64)      0
block2_conv1 (Conv2D)     (None, 112, 112, 128)     73856
```

block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

=====
Total params: 143667240 (548.05 MB)
Trainable params: 143667240 (548.05 MB)
Non-trainable params: 0 (0.00 Byte)

Vamos descartar as duas últimas camadas do VGG

```
# https://www.tensorflow.org/guide/keras/functional?hl=pt_br
from tensorflow.keras.models import Model

vgg_face_descriptor = Model(inputs=vgg19.layers[0].input, outputs=vgg19.layers[-2].output)
```

Responda

To Do - Por que descartamos as duas últimas camadas do VGG?

▼ 3.3 Medindo Similaridade

▼ As funções abaixo servem para medir similaridade entre duas imagens, passando-se um vetor de características.

```
def findCosineSimilarity(source_representation, test_representation):
    a = np.matmul(np.transpose(source_representation), test_representation)
    b = np.sum(np.multiply(source_representation, source_representation))
    c = np.sum(np.multiply(test_representation, test_representation))
    return 1 - (a / (np.sqrt(b) * np.sqrt(c)))

def findEuclideanDistance(source_representation, test_representation):
    euclidean_distance = source_representation - test_representation
    euclidean_distance = np.sum(np.multiply(euclidean_distance, euclidean_distance))
    euclidean_distance = np.sqrt(euclidean_distance)
    return euclidean_distance
```

▼ A função verifyFace recebe duas imagens e calcula a similaridade entre elas. Se a similaridade for menor que epsilon, afirma-se que as duas imagens são de uma mesma pessoa.

```

epsilon = 0.0040

def verifyFace(img1, img2):

    img1_representation = vgg_face_descriptor.predict(img1, steps=None)[0,:]
    img2_representation = vgg_face_descriptor.predict(img2, steps=None)[0,:]

    cosine_similarity = findCosineSimilarity(img1_representation, img2_representation)
    euclidean_distance = findEuclideanDistance(img1_representation, img2_representation)

    print("Similaridade com distancia do cosseno: ",cosine_similarity)
    print("Similaridade com distancia euclídea: ",euclidean_distance)

    if(cosine_similarity < epsilon):
        print("Verificado! Mesma pessoa!")
    else:
        print("Não-verificado! Não são a mesma pessoa!")

    f = plt.figure()
    f.add_subplot(1,2, 1)
    plt.imshow(np.squeeze(img1))
    plt.xticks([]); plt.yticks([])
    f.add_subplot(1,2, 2)
    plt.imshow(np.squeeze(img2))
    plt.xticks([]); plt.yticks([])
    plt.show(block=True)
    print("-----")

```

▼ Verificando a similaridade entre imagens (6pt)

Para 4 pares de imagens da base da AT&T e faça uma verificação entre elas, chamando a função verifyFace().

Antes de usar o VGG como um extrator de características, normalize os dados dividindo os pixels por 255. Além disso, re-escalone as imagens para o formato 224x224. Use a biblioteca OpenCV (cv2).

Faça para os pares : 64 e 33, 3 e 7, 40 e 44, 100 e 200.

```

import cv2

# Ajuste as imagens para a entrada do modelo VGG

# exemplo, para o par 64 e 33 :

array = [(64, 33), (3, 7), (40, 44), (100, 200)]

for index1, index2 in array:
    # Todo : Normaliza entre 0 e 1 , dividindo por 255
    img1 = X[index1,:,:,:].astype('float32') / 255.0
    img2 = X[index2,:,:,:].astype('float32') / 255.0

    # Redimensione a imagem para (224,224) e coloca a primeira dimensão unitária
    img1 = cv2.resize(img1, (224,224))
    img2 = cv2.resize(img2, (224,224))

    # lembre-se de expandir a primeira dimensão, pois nosso lote aqui é de 1 imagem
    img1 = np.expand_dims(img1, axis=0)
    img2 = np.expand_dims(img2, axis=0)

    verifyFace(img1, img2)

```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7cc111b64c10>
1/1 [=====] - 1s 859ms/step
1/1 [=====] - 0s 25ms/step
Similaridade com distancia do coseno:  0.0
Similaridade com distancia euclideana:  0.0
Verificado! Mesma pessoa!
```



```
-----[=====] - 0s 29ms/step
1/1 [=====] - 0s 20ms/step
Similaridade com distancia do coseno:  0.0
Similaridade com distancia euclideana:  0.0
Verificado! Mesma pessoa!
```



```
-----[=====] - 0s 31ms/step
1/1 [=====] - 0s 17ms/step
Similaridade com distancia do coseno:  0.0
Similaridade com distancia euclideana:  0.0
Verificado! Mesma pessoa!
```



4. Transferência de aprendizado (50pt)

Estude o tutorial do [link](#) e aplique o mesmo procedimento para ajustar um modelo previamente treinado com imagens da imagenet. Use o MobileNetV2 como modelo base.

Faça o procedimento em duas etapas:

1. Congele todas as camadas exceto as novas que você adicionou ao modelo. Treine.
2. Libere todas as camadas para o treinamento e treine novamente com um Learning Rate bem pequeno (um décimo do realizado no ítem 1).



```
# Usando o mobileNet, as imagens devem ter entrada de 160x160x3 e normalizadas entre 0 e 1.
# Use a função abaixo para fazer o trabalho, conjuntamente com tf.data.Dataset.from_tensor_slices
```

```
IMG_SIZE = 160 # All images will be resized to 160x160
```

```
def format_example(image, label):
    image = tf.cast(image, tf.float32)
    image = (image/127.5) - 1
```

```

image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
return image, label

X_train.shape
(280, 112, 92, 3)

# Tensorflow tem funções específicas para carregar os dados. Veja tf.data.Dataset

raw_train = tf.data.Dataset.from_tensor_slices((X_train,y_train))
raw_test = tf.data.Dataset.from_tensor_slices((X_test,y_test))

train = raw_train.map(format_example)
test = raw_test.map(format_example)

```

Seus dados devem ter o formato :

```
TensorShape([Dimension(280), Dimension(160), Dimension(160), Dimension(3)])
```

```

SHUFFLE_BUFFER_SIZE = 32
BATCH_SIZE = 32

train_batches = train.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE)
test_batches = test.batch(BATCH_SIZE)

```

▼ 4.1. Execute os passos (35pt):

1. Carregue o modelo pré-treinado do MobileNet, remova a última camada.
2. Adicione uma camada de Global Average Pooling 2D (GAP)
3. Adicione uma camada densa para ajustar ao seu número de classes e use ativação softmax
4. Use função de custo loss='sparse_categorical_crossentropy'
5. Dentre os dados de treinamento, reserve 10% para validação do modelo.
6. Treine por 10 épocas.
7. Pinte os gráficos de custo do treino e validação

```

from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense

IMG_SHAPE = (IMG_SIZE, IMG_SIZE) + (3,)

base_model = MobileNet(input_shape=IMG_SHAPE,
                       include_top=False,
                       weights='imagenet')

base_model.trainable = False

global_average_layer = GlobalAveragePooling2D()
feature_batch_average = global_average_layer(base_model.output)

prediction_layer = Dense(40, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)

inputs = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(inputs, training=False)
x = global_average_layer(x)
outputs = prediction_layer(x)
model = Model(inputs, outputs)

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_batches,
                     epochs=10,
                     batch_size=BATCH_SIZE,
                     validation_data=test_batches)

plt.plot(history.history['accuracy'], label='accuracy')

```

```

plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_batches, verbose=2)
test_acc

```

```

global_average_pooling2d ( (None, 1024)          0
                           GlobalAveragePooling2D)

dense_3 (Dense)           (None, 40)        41000

```

```

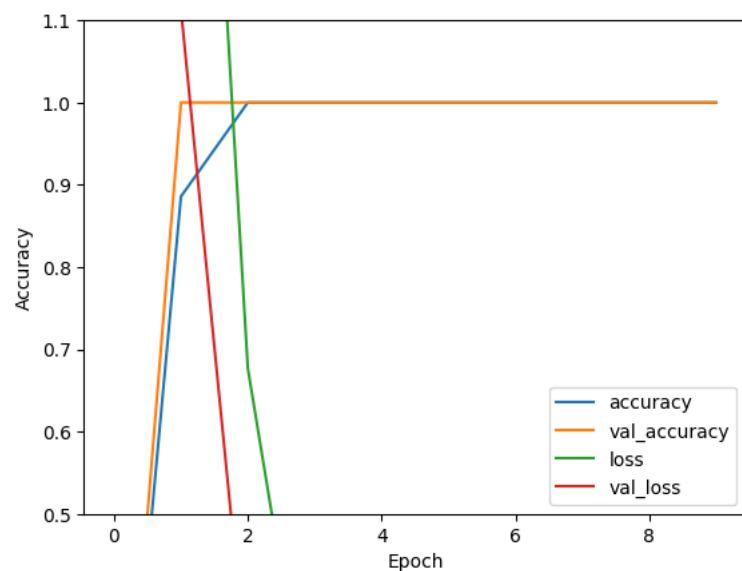
=====
Total params: 3269864 (12.47 MB)
Trainable params: 41000 (160.16 KB)
Non-trainable params: 3228864 (12.32 MB)

```

```

Epoch 1/10
9/9 [=====] - 3s 126ms/step - loss: 4.0130 - accuracy: 0.0000e+00 - val_loss: 2.8483 - val_accuracy: 0.000
Epoch 2/10
9/9 [=====] - 0s 42ms/step - loss: 2.0458 - accuracy: 0.8857 - val_loss: 1.1131 - val_accuracy: 1.000
Epoch 3/10
9/9 [=====] - 0s 35ms/step - loss: 0.6767 - accuracy: 1.0000 - val_loss: 0.2920 - val_accuracy: 1.000
Epoch 4/10
9/9 [=====] - 0s 42ms/step - loss: 0.1829 - accuracy: 1.0000 - val_loss: 0.0955 - val_accuracy: 1.000
Epoch 5/10
9/9 [=====] - 0s 34ms/step - loss: 0.0703 - accuracy: 1.0000 - val_loss: 0.0481 - val_accuracy: 1.000
Epoch 6/10
9/9 [=====] - 0s 43ms/step - loss: 0.0401 - accuracy: 1.0000 - val_loss: 0.0323 - val_accuracy: 1.000
Epoch 7/10
9/9 [=====] - 0s 46ms/step - loss: 0.0289 - accuracy: 1.0000 - val_loss: 0.0253 - val_accuracy: 1.000
Epoch 8/10
9/9 [=====] - 0s 45ms/step - loss: 0.0235 - accuracy: 1.0000 - val_loss: 0.0215 - val_accuracy: 1.000
Epoch 9/10
9/9 [=====] - 0s 41ms/step - loss: 0.0203 - accuracy: 1.0000 - val_loss: 0.0189 - val_accuracy: 1.000
Epoch 10/10
9/9 [=====] - 0s 40ms/step - loss: 0.0181 - accuracy: 1.0000 - val_loss: 0.0170 - val_accuracy: 1.000
4/4 - 0s - loss: 0.0170 - accuracy: 1.0000 - 108ms/epoch - 27ms/step
1.0

```



▼ 4.2. Fazendo testes (13pt)

Clique duas vezes (ou pressione "Enter") para editar

Analice os gráficos. Você provavelmente deve ter observado overfitting. Aplique algumas regularizações no modelo, para tentar reduzir o super-ajuste.

1. Dropout, antes da camada densa, de 50%
2. Regularização nos pesos da camada densa (L1 ou L2)
3. Dropout antes da camada de GAP

Veja exemplos no [link](#)

#Dropout

```
inputs = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(inputs, training=False)
x = global_average_layer(x)
x = Dropout(0.5)(x)
outputs = prediction_layer(x)
model = Model(inputs, outputs)

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_batches,
                     epochs=10,
                     batch_size=BATCH_SIZE,
                     validation_data=test_batches)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_batches, verbose=2)
test_acc
```

```

dropout (Dropout)           (None, 1024)          0
dense_3 (Dense)            (None, 40)           41000
=====
Total params: 3269864 (12.47 MB)
Trainable params: 41000 (160.16 KB)
Non-trainable params: 3228864 (12.32 MB)

Epoch 1/10
9/9 [=====] - 4s 101ms/step - loss: 0.0189 - accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 2/10
9/9 [=====] - 0s 45ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 6.9582e-04 - val_accuracy: 1.0000
Epoch 3/10
9/9 [=====] - 0s 43ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 3.1347e-04 - val_accuracy: 1.0000
Epoch 4/10
9/9 [=====] - 0s 35ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 1.9477e-04 - val_accuracy: 1.0000
Epoch 5/10
9/9 [=====] - 0s 35ms/step - loss: 7.9018e-04 - accuracy: 1.0000 - val_loss: 1.4435e-04 - val_accuracy: 1.
Epoch 6/10
9/9 [=====] - 0s 44ms/step - loss: 4.9854e-04 - accuracy: 1.0000 - val_loss: 1.1730e-04 - val_accuracy: 1.
Epoch 7/10
9/9 [=====] - 0s 36ms/step - loss: 3.8959e-04 - accuracy: 1.0000 - val_loss: 1.0049e-04 - val_accuracy: 1.
Epoch 8/10
9/9 [=====] - 0s 42ms/step - loss: 4.8893e-04 - accuracy: 1.0000 - val_loss: 8.8569e-05 - val_accuracy: 1.
Epoch 9/10
9/9 [=====] - 0s 35ms/step - loss: 4.0166e-04 - accuracy: 1.0000 - val_loss: 7.8317e-05 - val_accuracy: 1.
Epoch 10/10
    100%|██████████| 10/10 [00:00<00:00, 10.00s/epoch, loss=4.0166e-04, acc=1.0000, val_loss=7.8317e-05, val_acc=1.0000]

#L1 ou L2

regularizer_layer = tf.keras.layers.Dense(40, activation="softmax", kernel_regularizer=tf.keras.regularizers.l2(0.001))
regularizer_batch = regularizer_layer(feature_batch_average)

inputs = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(inputs, training=False)
x = global_average_layer(x)
outputs = regularizer_layer(x)
model = Model(inputs, outputs)

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_batches,
                     epochs=10,
                     batch_size=BATCH_SIZE,
                     validation_data=test_batches)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_batches, verbose=2)
test_acc

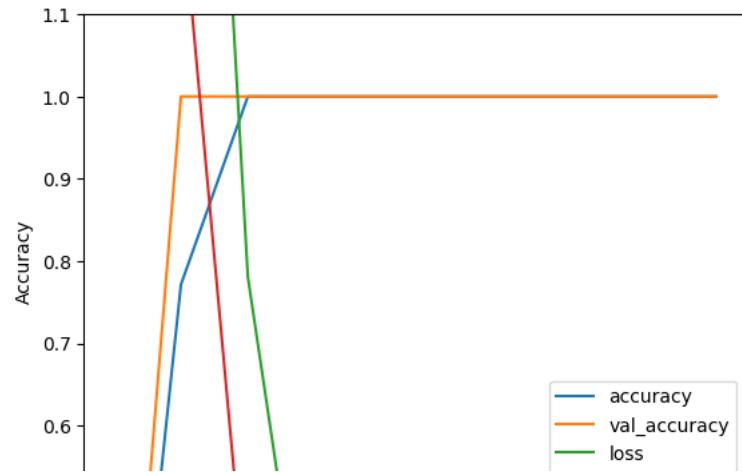
```

```
global_average_pooling2d ( (None, 1024)          0
                           GlobalAveragePooling2D)
```

```
dense_4 (Dense)      (None, 40)        41000
```

```
=====
Total params: 3269864 (12.47 MB)
Trainable params: 41000 (160.16 KB)
Non-trainable params: 3228864 (12.32 MB)
```

```
Epoch 1/10
9/9 [=====] - 2s 100ms/step - loss: 4.2154 - accuracy: 0.0000e+00 - val_loss: 3.0360 - val_accuracy: 0.000
Epoch 2/10
9/9 [=====] - 0s 38ms/step - loss: 2.2154 - accuracy: 0.7714 - val_loss: 1.2498 - val_accuracy: 1.0000
Epoch 3/10
9/9 [=====] - 0s 34ms/step - loss: 0.7808 - accuracy: 1.0000 - val_loss: 0.3593 - val_accuracy: 1.0000
Epoch 4/10
9/9 [=====] - 0s 44ms/step - loss: 0.2362 - accuracy: 1.0000 - val_loss: 0.1368 - val_accuracy: 1.0000
Epoch 5/10
9/9 [=====] - 0s 43ms/step - loss: 0.1075 - accuracy: 1.0000 - val_loss: 0.0815 - val_accuracy: 1.0000
Epoch 6/10
9/9 [=====] - 0s 49ms/step - loss: 0.0716 - accuracy: 1.0000 - val_loss: 0.0619 - val_accuracy: 1.0000
Epoch 7/10
9/9 [=====] - 0s 42ms/step - loss: 0.0574 - accuracy: 1.0000 - val_loss: 0.0527 - val_accuracy: 1.0000
Epoch 8/10
9/9 [=====] - 0s 49ms/step - loss: 0.0501 - accuracy: 1.0000 - val_loss: 0.0473 - val_accuracy: 1.0000
Epoch 9/10
9/9 [=====] - 0s 47ms/step - loss: 0.0457 - accuracy: 1.0000 - val_loss: 0.0438 - val_accuracy: 1.0000
Epoch 10/10
9/9 [=====] - 0s 46ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss: 0.0411 - val_accuracy: 1.0000
4/4 - 0s - loss: 0.0411 - accuracy: 1.0000 - 117ms/epoch - 29ms/step
1.0
```



#Dropout antes da GAP

```
inputs = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(inputs, training=False)
x = Dropout(0.5)(x)
x = global_average_layer(x)
outputs = prediction_layer(x)
model = Model(inputs, outputs)

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_batches,
                     epochs=10,
                     batch_size=BATCH_SIZE,
                     validation_data=test_batches)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
```

```

plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_batches, verbose=2)
test_acc

```

Model: "model_4"

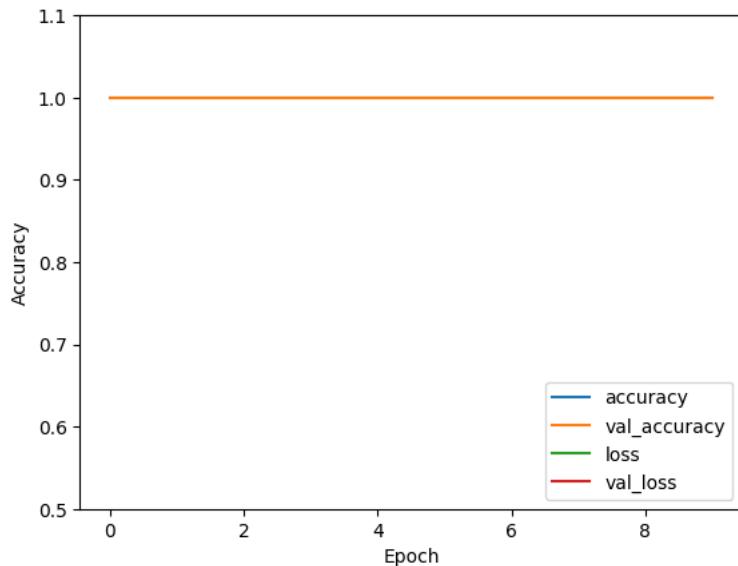
Layer (type)	Output Shape	Param #
input_7 (InputLayer)	[(None, 160, 160, 3)]	0
mobilenet_1.00_160 (Functional)	(None, 5, 5, 1024)	3228864
dropout_1 (Dropout)	(None, 5, 5, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dense_3 (Dense)	(None, 40)	41000

Total params: 3269864 (12.47 MB)
Trainable params: 41000 (160.16 KB)
Non-trainable params: 3228864 (12.32 MB)

```

Epoch 1/10
9/9 [=====] - 3s 107ms/step - loss: 4.5531e-05 - accuracy: 1.0000 - val_loss: 1.5378e-05 - val_accuracy: 1.
Epoch 2/10
9/9 [=====] - 0s 36ms/step - loss: 1.1576e-05 - accuracy: 1.0000 - val_loss: 5.0068e-06 - val_accuracy: 1.
Epoch 3/10
9/9 [=====] - 0s 34ms/step - loss: 4.5802e-06 - accuracy: 1.0000 - val_loss: 2.7418e-06 - val_accuracy: 1.
Epoch 4/10
9/9 [=====] - 0s 42ms/step - loss: 2.5885e-06 - accuracy: 1.0000 - val_loss: 1.6689e-06 - val_accuracy: 1.
Epoch 5/10
9/9 [=====] - 0s 37ms/step - loss: 1.7588e-06 - accuracy: 1.0000 - val_loss: 1.1921e-06 - val_accuracy: 1.
Epoch 6/10
9/9 [=====] - 0s 34ms/step - loss: 1.4867e-06 - accuracy: 1.0000 - val_loss: 8.3446e-07 - val_accuracy: 1.
Epoch 7/10
9/9 [=====] - 0s 34ms/step - loss: 1.1976e-06 - accuracy: 1.0000 - val_loss: 7.1526e-07 - val_accuracy: 1.
Epoch 8/10
9/9 [=====] - 1s 55ms/step - loss: 1.0614e-06 - accuracy: 1.0000 - val_loss: 7.1526e-07 - val_accuracy: 1.
Epoch 9/10
9/9 [=====] - 1s 49ms/step - loss: 9.2259e-07 - accuracy: 1.0000 - val_loss: 5.9605e-07 - val_accuracy: 1.
Epoch 10/10
9/9 [=====] - 0s 42ms/step - loss: 8.2467e-07 - accuracy: 1.0000 - val_loss: 5.9605e-07 - val_accuracy: 1.
4/4 - 0s - loss: 5.9605e-07 - accuracy: 1.0000 - 95ms/epoch - 24ms/step
1.0

```



Responda (2pt)

ToDo - com qual configuração conseguiu resolver o overfitting?