

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')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.remount.

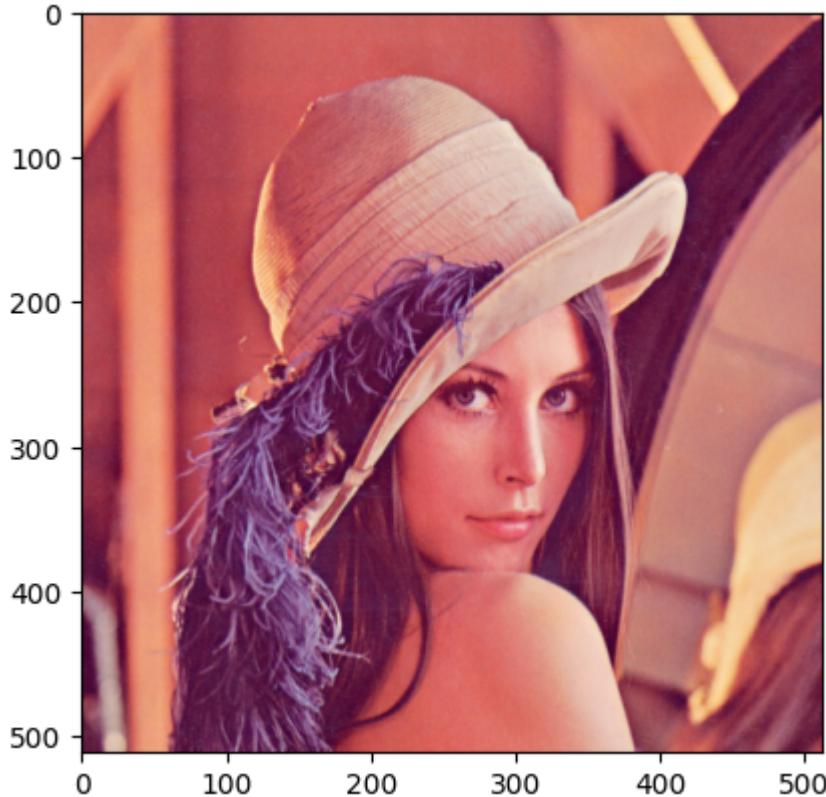
```
%matplotlib inline  
import matplotlib.pyplot as plt  
import numpy as np  
  
from skimage.io import imread  
from skimage.transform import resize
```

▼ 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.lay
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_7"
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_7 (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, dim
# Uma imagem isolada é considerada um lote de tamanho 1, portanto, deve-se expandir mais
img_in = np.expand_dims(sample_image, 0)
img_in.shape
```

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

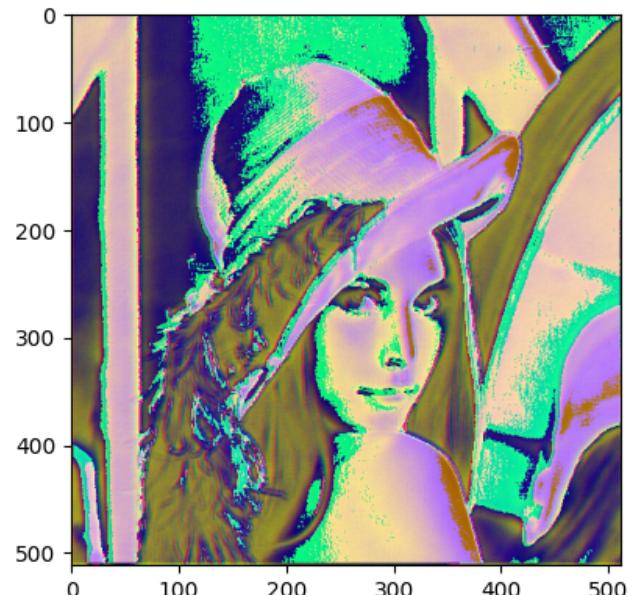
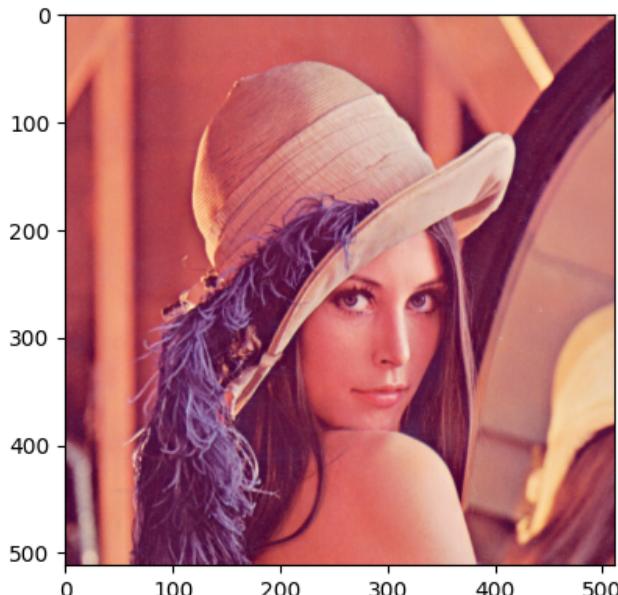
Agora, pode-se aplicar a convolução. Aplique a convolução na imagem de exemplo (expandida) e verifique o tamanho da imagem resultante (img_out). Use a função predict do objeto conv para aplicar a convolução.

```
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.
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, None, None, 1)	76

```
=====
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

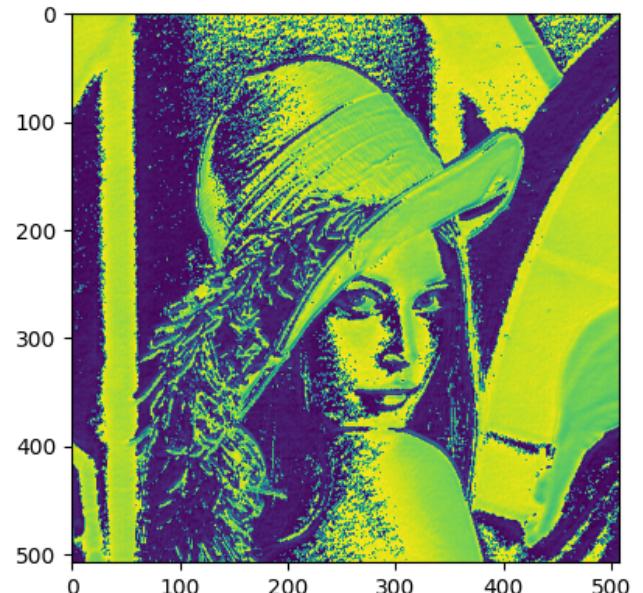
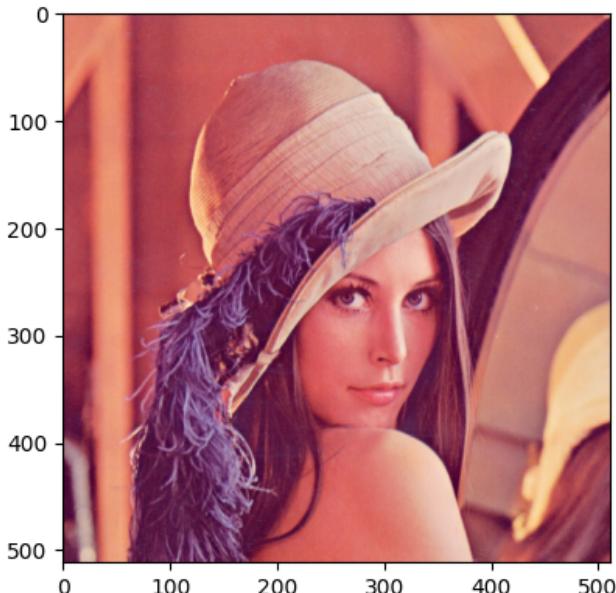
TensorShape([508, 508, 1])
```

Plote as duas imagens lado a lado

```
# Como tivemos que expandir a primeira dimensão para aplicar a convolução, podemos remov
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

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.]],

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

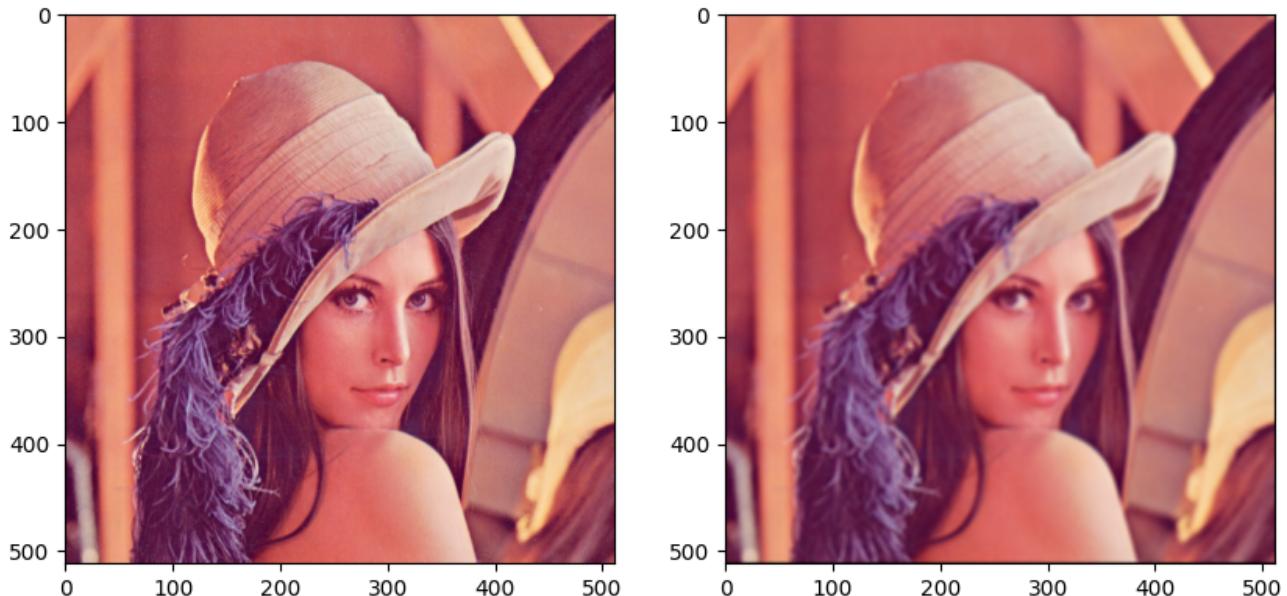
(None, None, None, 3)
```

▼ 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'));
```

WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function
1/1 [=====] - 0s 196ms/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):
    array1 = np.zeros(shape=shape, dtype=np.float32)
    array1 = [[0, 0.2, 0], [0, -0.2, 0], [0, 0, 0]]
    return array1
```

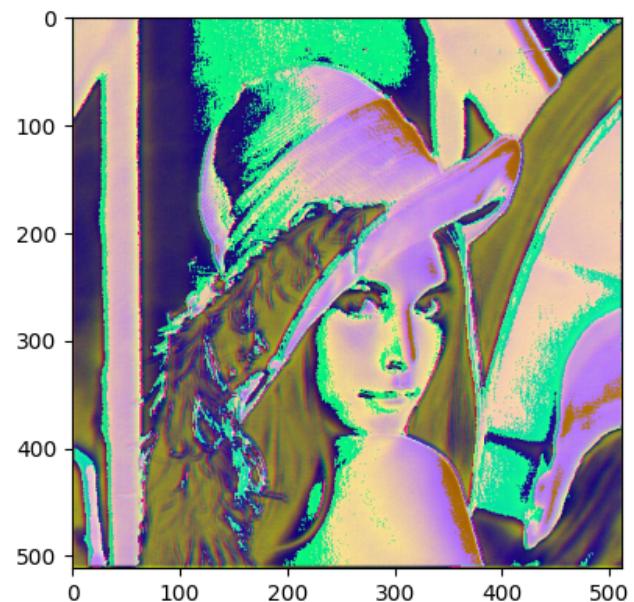
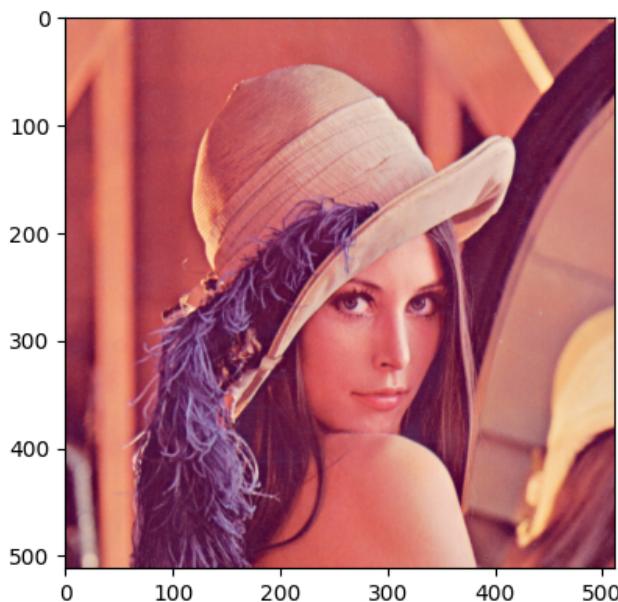
Inicialize o objeto conv4 com seu novo filtro e aplique na imagem de entrada

```
conv4 = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3), kernel_initializer=my_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 184ms/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.MaxPooling2D com kernel_size=2, 2 e stride=2
# lembre-se de colocar o parametro input_shape como input_shape=(None, None, 3)

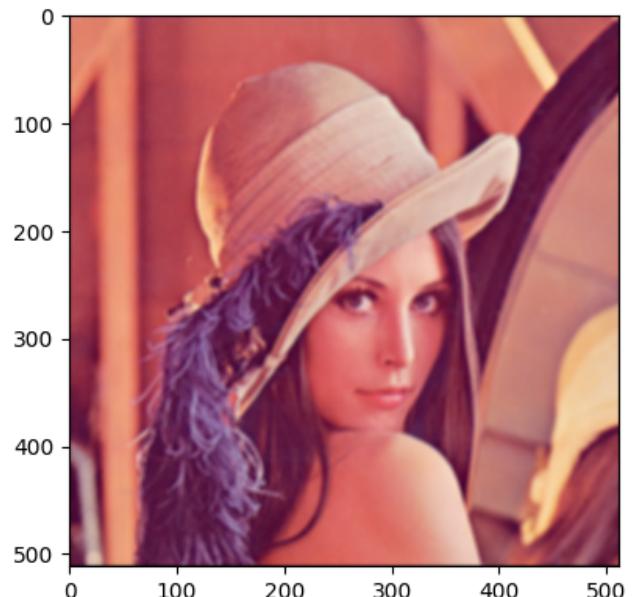
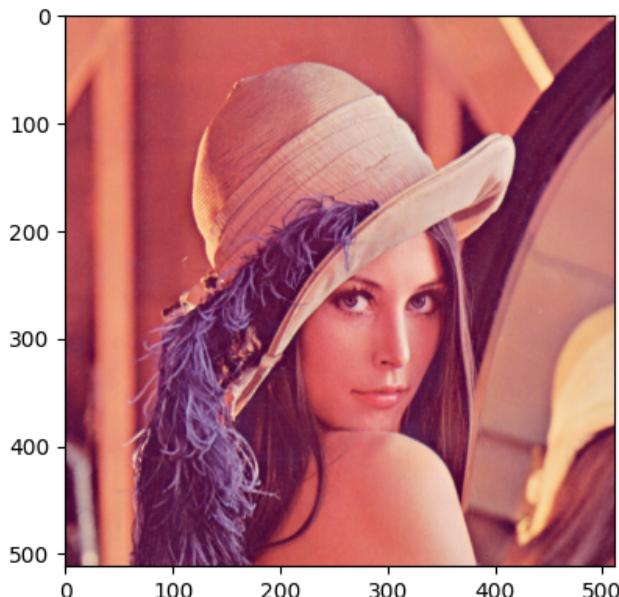
#-----rever
max_pool = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3), kernel_initializer=my_filter)
])
max_pool.output_shape

(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 211ms/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.layers
# lembre-se de colocar o parametro input-shape como input_shape=(None, None, 3)
# Coloque o parametro stride para 4
max_pool2 = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3), kernel_initializer=my_filter, strides = 4)
])

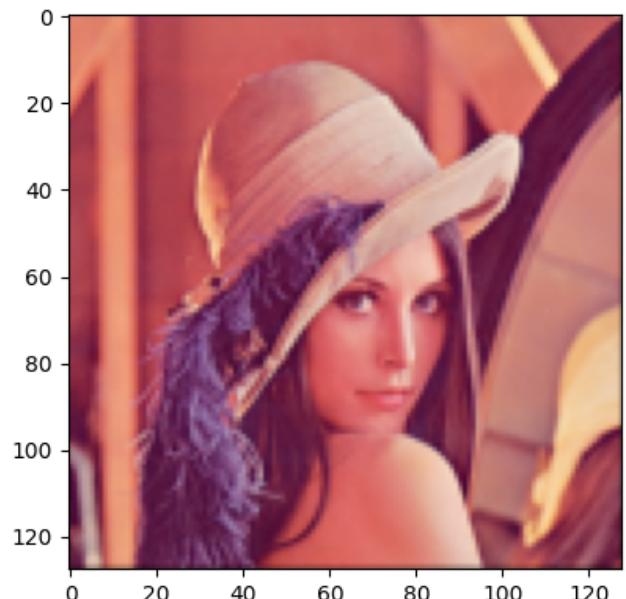
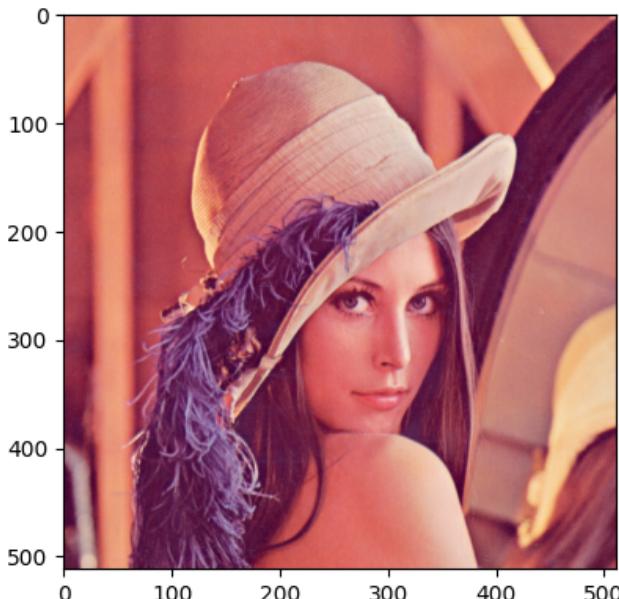
max_pool2.output_shape
```

(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 92ms/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.layers
# lembre-se de colocar o parametro input-shape como input_shape=(None, None, 3)

max_pool3 = Sequential([
    Conv2D(filters=3, kernel_size=(5, 5), padding="same",
           input_shape=(None, None, 3), kernel_initializer=my_filter, strides = 8)
])

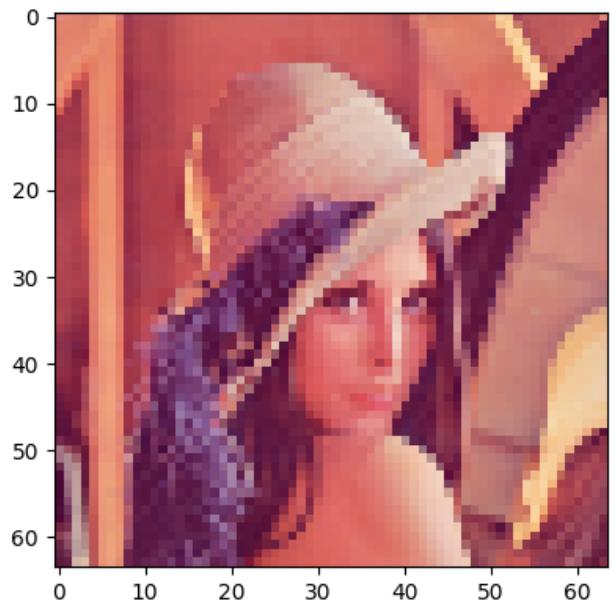
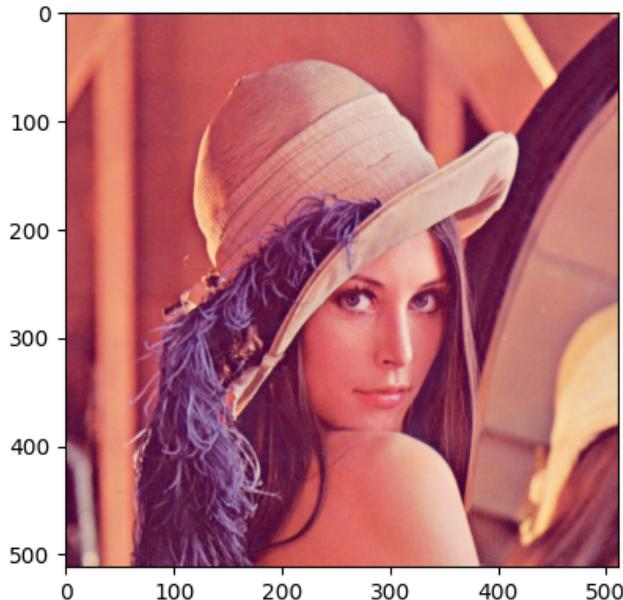
max_pool3.output_shape
```

(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
```

```
1/1 [=====] - 0s 65ms/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

To Do - Descreva o que aconteceu com o aumento do stride.

Com o aumento do stride podemos observar que a imagem fica mais pixelada, perdendo assim alguns detalhes, mas ainda é possível reconhecer que é a mesma imagem.

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

O objetivo desta etapa é classificar faces na base ORL (AT&T) Database (40 indivíduos 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)

```
# carregue as imagens

# 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
 = "/content/drive/My Drive/datasets/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)

# Divida 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 = 0.70,train_size = 0.30)

X_train.shape

(120, 112, 92)

X_test.shape

(280, 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)) # ToDo ...
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# ToDo : adicionar as outras camadas

# Segunda camada de convolução
model.add(layers.Conv2D(64, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Terceira camada de convolução
model.add(layers.Conv2D(96, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten()) # não esqueça da camada flatten ..

model.add(layers.Dense(128, activation='relu')) # Todo ..
model.add(layers.Dense(100, activation='relu')) # Todo: softmax

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

model.summary()
```

Model: "sequential_14"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_14 (Conv2D)	(None, 110, 90, 32)	320
max_pooling2d (MaxPooling2D)	(None, 55, 45, 32)	0
conv2d_15 (Conv2D)	(None, 51, 41, 64)	51264
max_pooling2d_1 (MaxPooling2D)	(None, 25, 20, 64)	0
conv2d_16 (Conv2D)	(None, 23, 18, 96)	55392
max_pooling2d_2 (MaxPooling2D)	(None, 11, 9, 96)	0
flatten (Flatten)	(None, 9504)	0
dense (Dense)	(None, 128)	1216640

dense_1 (Dense)	(None, 100)	12900
dense_2 (Dense)	(None, 40)	4040
<hr/>		
Total params: 1340556 (5.11 MB)		
Trainable params: 1340556 (5.11 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 primeira dimensão é o tamanho do lote, a segunda e terceira
# Repare que as imagens desta base tem apenas uma banda (escala de cinza)
X_train.shape
```

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

```
# Como o tensor acima não contempla o tamanho de canais (no caso , igual a 1), deve-se ex
X_train_new = np.expand_dims(X_train, axis=-1)
X_test_new = np.expand_dims(X_test, axis=-1)

X_train_new.shape

(120, 112, 92, 1)

# o vetor de rótulos não precisa ter duas dimensões.
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;

print(y_train_new.dtype)
print(y_train_new.shape)
print(np.min(y_train_new), np.max(y_train_new))

float64
(120,)
0.0 39.0
```

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.

```
y_train_new = y_train_new.astype(int)
y_test_new = y_test_new.astype(int)

# 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=
```

2/2 [=====] - 8s 6s/step - loss: 29.9738 - accuracy: 0.00
Epoch 3/30
2/2 [=====] - 8s 6s/step - loss: 7.6479 - accuracy: 0.033
Epoch 4/30
2/2 [=====] - 8s 6s/step - loss: 7.8881 - accuracy: 0.033
Epoch 5/30
2/2 [=====] - 5s 3s/step - loss: 5.7064 - accuracy: 0.016
Epoch 6/30

```
[  0%] - 5s 3s/step - loss: 3.6154 - accuracy: 0.075 ▲  
Epoch 9/30  
[  2%] - 6s 3s/step - loss: 3.5841 - accuracy: 0.116  
Epoch 10/30  
[  4%] - 7s 3s/step - loss: 3.5098 - accuracy: 0.116  
Epoch 11/30  
[  6%] - 5s 2s/step - loss: 3.3747 - accuracy: 0.183  
Epoch 12/30  
[  8%] - 4s 2s/step - loss: 3.3044 - accuracy: 0.158  
Epoch 13/30  
[ 10%] - 9s 6s/step - loss: 3.1829 - accuracy: 0.216  
Epoch 14/30  
[ 12%] - 5s 3s/step - loss: 3.1412 - accuracy: 0.233  
Epoch 15/30  
[ 14%] - 5s 3s/step - loss: 2.8595 - accuracy: 0.283  
Epoch 16/30  
[ 16%] - 10s 6s/step - loss: 2.5745 - accuracy: 0.32  
Epoch 17/30  
[ 18%] - 5s 3s/step - loss: 2.3676 - accuracy: 0.466  
Epoch 18/30  
[ 20%] - 7s 5s/step - loss: 2.0905 - accuracy: 0.483  
Epoch 19/30  
[ 22%] - 8s 2s/step - loss: 1.9232 - accuracy: 0.433  
Epoch 20/30  
[ 24%] - 5s 3s/step - loss: 1.5857 - accuracy: 0.566  
Epoch 21/30  
[ 26%] - 8s 6s/step - loss: 1.3189 - accuracy: 0.683  
Epoch 22/30  
[ 28%] - 5s 3s/step - loss: 1.1037 - accuracy: 0.741  
Epoch 23/30  
[ 30%] - 4s 2s/step - loss: 1.4947 - accuracy: 0.541  
Epoch 24/30  
[ 32%] - 9s 6s/step - loss: 0.9334 - accuracy: 0.750  
Epoch 25/30  
[ 34%] - 5s 3s/step - loss: 0.9118 - accuracy: 0.766  
Epoch 26/30  
[ 36%] - 12s 7s/step - loss: 0.8936 - accuracy: 0.75  
Epoch 27/30  
[ 38%] - 8s 3s/step - loss: 0.5985 - accuracy: 0.825  
Epoch 28/30  
[ 40%] - 8s 6s/step - loss: 0.4670 - accuracy: 0.875  
Epoch 29/30  
[ 42%] - 5s 3s/step - loss: 0.2597 - accuracy: 0.916  
Epoch 30/30  
[ 44%] - 4s 2s/step - loss: 0.2591 - accuracy: 0.941
```

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.

```

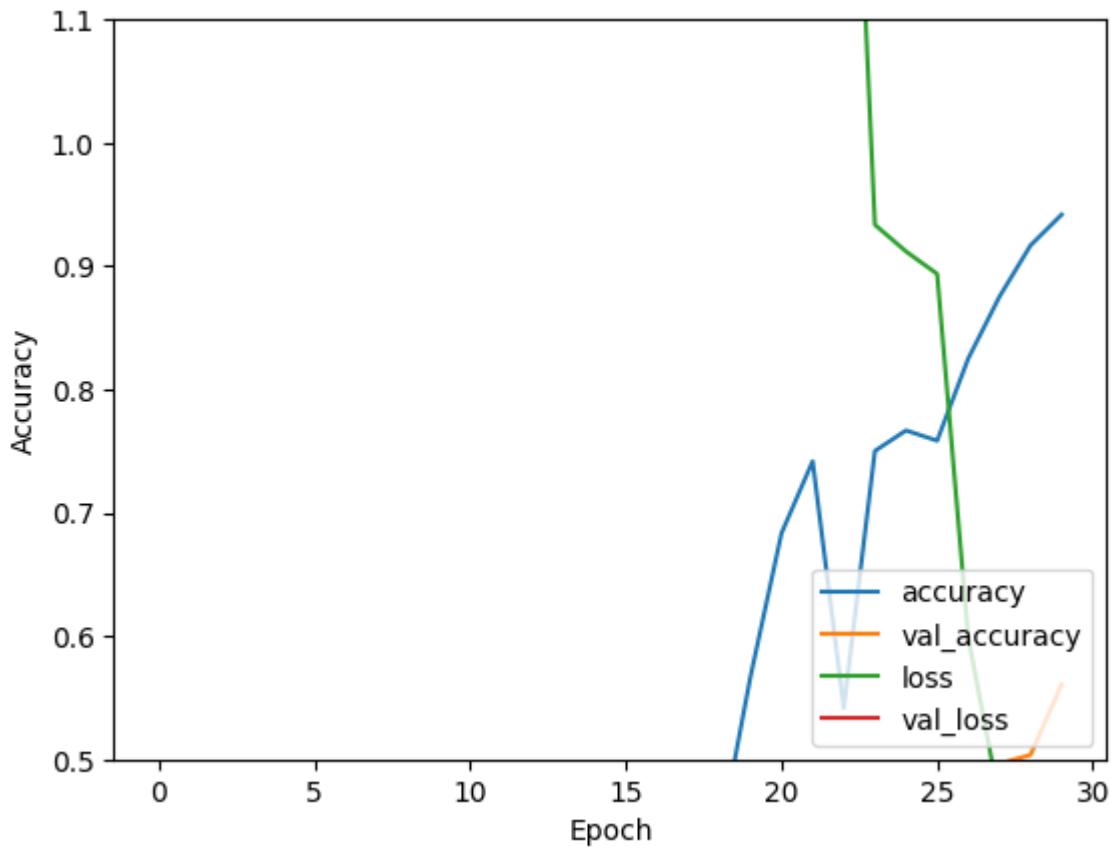
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.1])
plt.legend(loc='lower right')

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)

```

9/9 - 3s - loss: 2.2125 - accuracy: 0.5607 - 3s/epoch - 333ms/step



```
print(test_acc)
```

0.5607143044471741

▼ 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.

▼ 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/My Drive/datasets/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 = 0.70,train_size =
X_train.shape

(120, 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/vgg19/574710816/574710816 [=====] - 3s 0us/step
```

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

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_conv2 (Conv2D)	(None, 112, 112, 128)	147584
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

ToDo - Por que descartamos as duas últimas camadas do VGG?

Queremos aproveitar as camadas convolucionais para extração de características.

▼ 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 :

# Todo : Normaliza entre 0 e 1 , dividindo por 255
img1 = X[64,:,:,:] / 255.0 # Todo
img2 = X[33,:,:,:] / 255.0 # Todo

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

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

verifyFace(img1, img2)

```

```

1/1 [=====] - 2s 2s/step
1/1 [=====] - 2s 2s/step
Similaridade com distancia do cosseno:  0.009015381336212158
Similaridade com distancia euclideana:  6.17371
Não-verificado! Não são a 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 adicinou 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
# 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

(120, 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)])

```
train_batches = train.shuffle(32).batch(32)
test_batches = test.batch(32)
```

▼ 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. Plote os gráficos de custo do treino e validação

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

# Suponha que você tenha seus dados de entrada e rótulos (X e y)
# Substitua isso pelos seus dados reais
#X = ...
#y = ...

# Carregue o modelo pré-treinado do MobileNet sem a última camada
base_model = MobileNet(weights='imagenet', include_top=False)

# Adicione uma camada de Global Average Pooling 2D (GAP)
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Adicione uma camada densa para ajustar ao número de classes
num_classes = len(np.unique(y)) # assumindo que seus rótulos são inteiros consecutivos
predictions = Dense(num_classes, activation='softmax')(x)

# Crie o modelo final
model = Model(inputs=base_model.input, outputs=predictions)

# Congele as camadas do modelo base (MobileNet)
for layer in base_model.layers:
    layer.trainable = False

# Compile o modelo
model.compile(optimizer=Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Divida os dados em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=42)

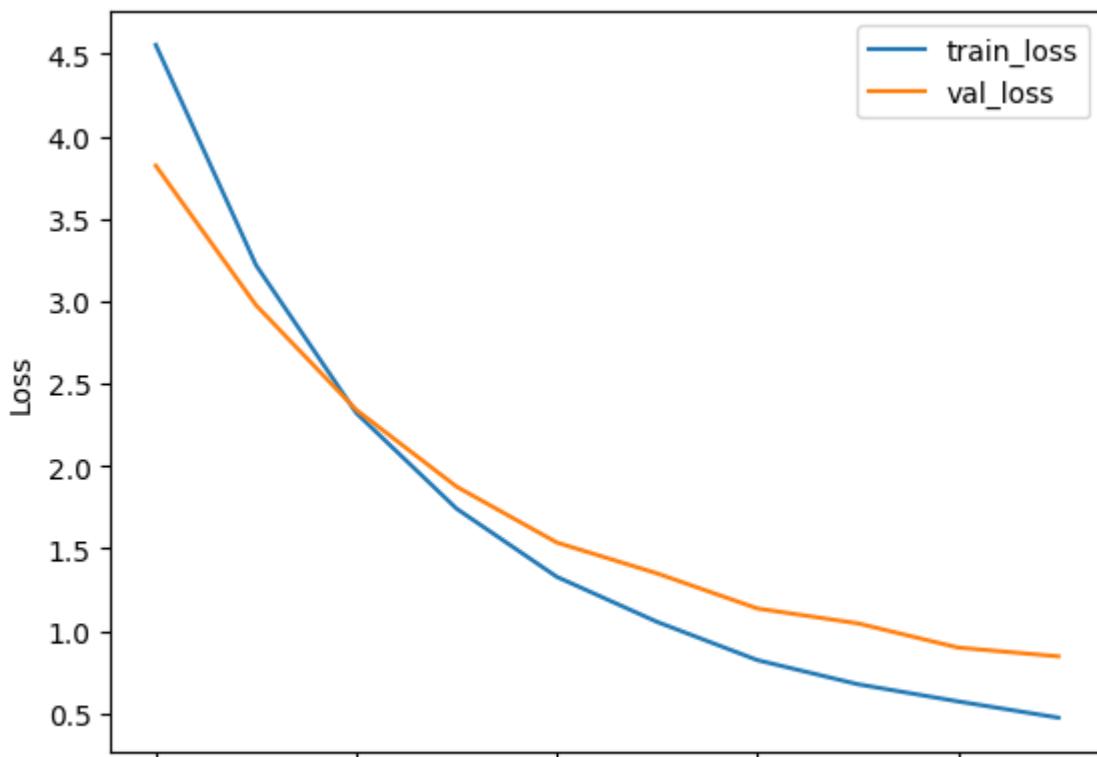
# Treine o modelo por 10 épocas
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Plote os gráficos de custo do treino e validação
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```

WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mo
17225924/17225924 [=====] - 0s 0us/step
Epoch 1/10
12/12 [=====] - 5s 245ms/step - loss: 4.5553 - accuracy: 0.0
Epoch 2/10
12/12 [=====] - 2s 188ms/step - loss: 3.2186 - accuracy: 0.1
Epoch 3/10
12/12 [=====] - 3s 232ms/step - loss: 2.3216 - accuracy: 0.4
Epoch 4/10
12/12 [=====] - 4s 324ms/step - loss: 1.7413 - accuracy: 0.6
Epoch 5/10
12/12 [=====] - 3s 238ms/step - loss: 1.3278 - accuracy: 0.7
Epoch 6/10
12/12 [=====] - 2s 176ms/step - loss: 1.0544 - accuracy: 0.8
Epoch 7/10
12/12 [=====] - 2s 185ms/step - loss: 0.8225 - accuracy: 0.9
Epoch 8/10
12/12 [=====] - 2s 179ms/step - loss: 0.6765 - accuracy: 0.9
Epoch 9/10
12/12 [=====] - 2s 183ms/step - loss: 0.5722 - accuracy: 0.9
Epoch 10/10
12/12 [=====] - 3s 290ms/step - loss: 0.4736 - accuracy: 0.9

```



▼ 4.2. Fazendo testes (13pt)

Analize 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](#)

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.model_selection import train_test_split

# Suponha que você tenha seus dados de entrada e rótulos (X e y)
# Substitua isso pelos seus dados reais
#X = ...
#y = ...

# Carregue o modelo pré-treinado do MobileNet sem a última camada
base_model = MobileNet(weights='imagenet', include_top=False)

# Adicione uma camada de Global Average Pooling 2D (GAP)
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Adicione dropout antes da camada densa
x = Dropout(0.5)(x)

# Adicione uma camada densa para ajustar ao número de classes
num_classes = len(np.unique(y)) # assumindo que seus rótulos são inteiros consecutivos
x = Dense(num_classes, activation='softmax', kernel_regularizer=regularizers.l2(0.01))(x)

# Crie o modelo final
model = Model(inputs=base_model.input, outputs=x)

# Congele as camadas do modelo base (MobileNet)
for layer in base_model.layers:
    layer.trainable = False

# Compile o modelo
model.compile(optimizer=Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Divida os dados em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=42)

# Adicione dropout antes da camada de GAP
x_with_dropout = Dropout(0.5)(model.layers[-2].output) # exclua a camada de saída
predictions_with_dropout = Dense(num_classes, activation='softmax')(x_with_dropout)

# Crie o modelo final com dropout
model_with_dropout = Model(inputs=model.input, outputs=predictions_with_dropout)

# Compile o modelo com dropout
model_with_dropout.compile(optimizer=Adam(),
```

```
loss= sparse_categorical_crossentropy ,  
metrics=['accuracy'])  
  
# Treine o modelo por 10 épocas  
history_with_dropout = model_with_dropout.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))  
  
# Plote os gráficos de custo do treino e validação  
plt.plot(history_with_dropout.history['loss'], label='train_loss_with_dropout')  
plt.plot(history_with_dropout.history['val_loss'], label='val_loss_with_dropout')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()
```

WARNING:tensorflow: `input_shape` is undefined or non-square, or `rows` is not in [128, 64, 32].
Epoch 1/10
12/12 [=====] - 6s 389ms/step - loss: 9.6666 - accuracy: 0.0
Epoch 2/10
12/12 [=====] - 3s 291ms/step - loss: 8.4483 - accuracy: 0.0
Epoch 3/10
12/12 [=====] - 2s 185ms/step - loss: 7.7423 - accuracy: 0.0
Epoch 4/10
12/12 [=====] - 2s 187ms/step - loss: 7.3729 - accuracy: 0.0
Epoch 5/10
12/12 [=====] - 2s 177ms/step - loss: 6.5896 - accuracy: 0.0
Epoch 6/10
12/12 [=====] - 2s 177ms/step - loss: 5.8781 - accuracy: 0.1
Epoch 7/10
12/12 [=====] - 3s 247ms/step - loss: 5.8395 - accuracy: 0.1
Epoch 8/10
12/12 [=====] - 4s 320ms/step - loss: 5.2895 - accuracy: 0.1
Epoch 9/10
12/12 [=====] - 3s 229ms/step - loss: 4.7749 - accuracy: 0.1
Epoch 10/10
12/12 [=====] - 2s 187ms/step - loss: 4.8291 - accuracy: 0.1

10