

▼ Trabalho #7 - Reconhecimento de face

Nesse trabalho você vai criar um sistema de reconhecimento de face. O método utilizado nesse trabalho é da referência [FaceNet](#) que usa a função de custo tripla. Outro método visto em aula é o da referência [DeepFace](#), que usa classificação binária.

Os problemas de reconhecimento de rosto geralmente se enquadram em duas categorias:

- **Verificação de rosto** - "essa é a pessoa que diz ser?" Por exemplo, em alguns aeroportos, você pode passar pela alfândega deixando um sistema escanear seu passaporte e depois verificar se você (a pessoa que carrega o passaporte) é a pessoa correta. Um celular que desbloqueia usando seu rosto também está usando verificação de rosto. Este é um problema de correspondência 1 para 1.
- **Reconhecimento Facial** - "quem é essa pessoa?" Por exemplo, o vídeo mostrado na aula (<https://www.youtube.com/watch?v=wr4rx0Spihs>) de funcionários da Baidu que entraram no escritório sendo identificados pela face. Este é um problema de correspondência 1 para N.

A FaceNet é uma rede neural que codifica uma imagem de rosto em um vetor de 128 números. Ao comparar dois desses vetores, é possível determinar se duas imagens são da mesma pessoa.

Nesta tarefa, você irá:

- Implementar a função de custo tripla;
- Usar um modelo pré-treinado para mapear (codificar) imagens de rostos em vetores de 128 elementos;
- Usar essas codificações para executar a verificação e reconhecimento de faces.

Neste trabalho, usaremos um modelo pré-treinado que representa as ativações das camadas convolucionais usando a convenção do primeiro eixo ser dos canais (filtros), em oposição à convenção do último eixo ser dos canais como usada nas aulas e nos trabalhos anteriores. Em outras palavras, um lote de imagens tem a forma (m, n_C, n_H, n_W) em vez de (m, n_H, n_W, n_C) . Ambas as convenções têm uma quantidade razoável de aplicações entre implementações de código aberto, sendo que ainda não existe um padrão uniforme na área de deep learning.

Esse trabalho é adaptado de Andrew Ng (deeplearning.ai)

▼ Coloque os nomes e RAs dos alunos que fizeram esse trabalho

Nome e número dos alunos da equipe:

Aluno 1: Igor Amaral Correa 20.83992-8

Aluno 2:

Arquivos e subdiretórios necessários

Nesse trabalho você precisa de diversos arquivos:

1) Diretório com imagens

<https://drive.google.com/drive/folders/1B7sIDnrH0ZpiWvNzmjUTnwblfrOIM-7Z?usp=sharing>

2) Diretório com arquivos de parâmetros da rede

<https://drive.google.com/drive/folders/11bAqIL2QEiYepSFIZt70b9g2YWFram3h?usp=sharing>

3) Diretório com base de dados

<https://drive.google.com/drive/folders/1uGv0eNo7pUJQjR5W3imxoE8Tto2o24aG?usp=sharing>

Faça o download desses diretórios e coloque-os com o mesmo nome no diretório onde estão os arquivos `fr_util.py` e `inception_blocks_v2.py`.

Observa-se que a base de dados não é necessária para esse trabalho. Ela é necessária se você quiser retreinar a rede.


▼ Bibliotecas

Execute as células abaixo para importar as bibliotecas necessárias para o trabalho.

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

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
%cd /content/drive/My Drive/colab/RNC/T7/  
!ls
```

 /content/drive/My Drive/colab/RNC/T7
datasets fr_utils.py images inception_blocks_v2.py __pycache__ weights

```
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import BatchNormalization  
from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D  
from tensorflow.keras.layers import Lambda, Flatten, Dense  
from tensorflow.keras.initializers import glorot_uniform  
from tensorflow.keras.layers import Layer  
from tensorflow.keras import backend as K  
K.set_image_data_format('channels_first')
```

```
import cv2  
import os  
import numpy as np  
from numpy import genfromtxt  
import pandas as pd  
from fr_utils import *
```

```
%matplotlib inline  
%load_ext autoreload  
%autoreload 2
```

▼ 1 - Verificação de rosto ingênua

Na Verificação de rosto, você recebe duas imagens e precisa dizer se elas são da mesma pessoa. A maneira mais simples de tentar fazer isso seria comparar as duas imagens pixel por pixel. Se a distância entre as imagens for menor que o limite escolhido, pode ser a mesma pessoa!



Figura 1

Obviamente, esse algoritmo apresenta um desempenho muito ruim, pois os valores dos pixels mudam drasticamente devido a variações na iluminação, orientação do rosto da pessoa, alterações na posição da cabeça e assim por diante.

Vimos em aula que em vez de usar a imagem não processada, utiliza-se uma codificação $f(img)$ da imagem para que as comparações entre elementos dessa codificação forneçam julgamentos mais precisos sobre se duas imagens são da mesma pessoa.

▼ 2 - Codificando imagens de rosto em um vetor de 128 elementos

2.1 - Uso de uma rede convolucional para calcular as codificações

A rede FaceNet necessita de muitos dados e muito tempo para ser treinada. Portanto, seguindo a prática comum em desenvolvimento de redes deep learning, vamos utilizar uma rede pré-treinada. A arquitetura dessa rede segue o modelo de [Szegedy et al.] (<https://arxiv.org/abs/1409.4842>). Essa rede é do tipo "inception" e está configurada no arquivo `inception_blocks.py`. Dê um olhada no arquivo para ver como ela é implementada.

As principais coisas que você precisa saber a rede FaceNet são:

- Esta rede usa imagens RGB com resolução de 96x96x3 pixels como entrada. Especificamente, a entrada é uma imagem de rosto (ou um lote de m imagens de rostos) na forma de um tensor de dimensão $(m, n_C, n_H, n_W) = (m, 3, 96, 96)$;
- A saída dessa rede é uma matriz de dimensão $(m, 128)$ que codifica cada imagem de face em um vetor de 128 elementos.

Execute a célula abaixo para criar a rede que codifica imagens de rosto.

```
from inception_blocks_v2 import *
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
FRmodel.summary()
```

Note que essa rede possui cerca de 3,7 milhões de parâmetros.

Analise a configuração dessa rede e observe os seguintes pontos:

- As camadas convolucionais realizam o processo de "batch normalization";
- Essa rede possui vários módulos "inceptions";
- A última camada da rede é uma camada densa com 128 unidades, responsável por gerar na saída da rede um vetor de 128 elementos.

Ao usar uma camada totalmente conectada de 128 neurônios como última camada, a rede garante que a saída seja um vetor de codificação de 128 elementos. Esse vetor é então usado para comparar duas imagens de rosto. A Figura 2 mostra o processo utilizado para comparar duas imagens de rosto.



Figura 2

Ao calcular a distância entre duas codificações e comparando com um limiar, pode-se determinar se as duas imagens representam a mesma pessoa.

Portanto, uma codificação é boa se:

- As codificações de duas imagens de uma mesma pessoa são bastante semelhantes entre si;
- As codificações de duas imagens de pessoas diferentes são muito diferentes.

A função de custo tripla formaliza esse processo ao "aproximar" as codificações de duas imagens da mesma pessoa (referência e positiva) e "separar" as codificações de duas imagens de pessoas diferentes (referência e negativa).



Figura 3

2.2 - Função de custo tripla

Para uma imagem X , denotamos a sua codificação por $f(X)$, onde f é uma função calculada pela rede neural.



Figura 4

Após a codificação realizada pela rede neural, os vetores $f(X)$ são normalizados para ter norma igual a 1, ou seja:

$$\| f(x) \|_2 = 1$$

O treinamento da rede usa três imagens (R, P, N) :

- R é a imagem de "Referência" de uma pessoa;
- P é a imagem "Positiva", ou seja, uma imagem da mesma pessoa da imagem de "Referência";
- N é a imagem "Negativa", ou seja, uma imagem de uma pessoa diferente da imagem de "Referência".

Esses conjuntos de 3 imagens (triplos) são escolhidos dos exemplos de treinamento. Usaremos a notação $(R^{(i)}, P^{(i)}, N^{(i)})$ para representar o i -ésimo exemplo de treinamento, que consiste de 3 imagens.

Deseja-se garantir que a imagem $R^{(i)}$ de um indivíduo esteja mais próxima da imagem positiva $P^{(i)}$ do que da imagem negativa $N^{(i)}$ por pelo menos uma margem α . Esse critério é descrito pela seguinte equação:

$$\|f(R^{(i)}) - f(P^{(i)})\|_2^2 + \alpha < \|f(R^{(i)}) - f(N^{(i)})\|_2^2$$

Assim é desejado minimizar a seguinte função de custo tripla:

$$\mathcal{J} = \sum_{i=1}^m \max \left[\underbrace{\|f(R^{(i)}) - f(P^{(i)})\|_2^2}_{(1)} - \underbrace{\|f(R^{(i)}) - f(N^{(i)})\|_2^2}_{(2)} + \alpha, 0 \right]$$

Note que o valor da função de custo tripla é menor ou igual a zero, ou seja o seu valor máximo é zero.

Observações:

- O termo (1) da equação acima é a distância ao quadrado entre a imagem de referência R e a imagem positiva P para uma dada tripla. É desejado que esse valor seja pequeno;
- O termo (2) da equação acima é a distância ao quadrado entre a imagem de referência R e a negativa N para uma determinada tripla. É desejado que esse valor seja relativamente grande.
- α é a margem, que consiste de um hiperparâmetro que deve ser escolhido. Nesse trabalho usaremos $\alpha = 0.2$.

A maioria das implementações também normaliza os vetores de codificação para ter norma igual a um (isto é, $\|f(x)\|_2 = 1$). Não usaremos isso nesse trabalho.

▼ Exercício #1: Implementação da função de custo tripla

Os passos para implementar a função de custo tripla são os seguintes:

1. Calcule a distância entre as codificações das imagens de "referência" e "positiva": $\| f(R^{(i)}) - f(P^{(i)}) \|_2^2$;
2. Calcule a distância entre as codificações das imagens de "referência" e "negativa": $\| f(R^{(i)}) - f(N^{(i)}) \|_2^2$;
3. Para cada exemplo treinamento calcule a expressão: $\| f(R^{(i)}) - f(P^{(i)}) \|_2^2 - \| f(R^{(i)}) - f(N^{(i)}) \|_2^2 + \alpha$;
4. Calcule a expressão completa, tomando o máximo com zero e somando para todos os exemplos de treinamento.

Funções úteis: `tf.reduce_sum()`, `tf.square()`, `tf.subtract()`, `tf.add()`, `tf.maximum()`.

Nos passos 1 e 2, você tem que somar todos os elementos de $\| f(A^{(i)}) - f(P^{(i)}) \|_2^2$ e $\| f(A^{(i)}) - f(N^{(i)}) \|_2^2$, enquanto que no passo 4 você tem que somar para todos os exemplos de treinamento.

PARA VOCÊ FAZER: Função de custo tripla

```
def triplet_loss(y_true, y_pred, alpha = 0.2):
    """
    Argumentos:
    y_true = rótulo desejado, exigido quando se define um função de custo no Keras, mas nesse caso não é necessário
    y_pred = lista python contendo 3 objetos:
        Referência = codificação da imagem de referência, diomensão (None, 128)
        Positiva = codificação da imagem positivo, dimensão (None, 128)
        Negativa = codificação da imagem negativa, dimensão (None, 128)

    Retorna:
    loss = número real que consiste no valor da função de custo tripla
    """

    referencia, positiva, negativa = y_pred[0], y_pred[1], y_pred[2]

    ### COMECE AQUI ### (~ 4 linhas)
    # Passo 1: Calcule a distância entre as codificações da referência e positivathe, necessário somar ao longo de axis=-1
    pos_dist = tf.square( tf.subtract( referencia, positiva ) )
    # Passo 2: Calcule a distância entre as codificações da referência e negativa, necessário somar ao longo de axis=-1
    neg_dist = tf.square( tf.subtract( referencia, negativa ) )
    # Passo 3: subtrair as distâncias calculadas nos passos anteriores e somar alpha.
    basic_loss = tf.add( tf.subtract( pos_dist, neg_dist ), alpha )
    # Passo 4: Calcule o máximo entre basic_loss e 0.0 e depois some para todos es exemplos.
    loss = tf.reduce_sum( tf.maximum( basic_loss, 0.0 ) )
```

```
loss = tf.reduce_sum( tf.maximum( basic_loss, 0.0 ) ,  
### TERMINE AQUI ###
```

```
return loss
```

```
tf.random.set_seed(1)  
y_true = (None, None, None)  
y_pred = (tf.random.normal([3, 128], mean=6, stddev=0.1, seed = 1),  
          tf.random.normal([3, 128], mean=1, stddev=1, seed = 1),  
          tf.random.normal([3, 128], mean=3, stddev=4, seed = 1))  
loss = triplet_loss(y_true, y_pred)  
  
print("loss = ", format(loss))
```

 loss = 4882.20556640625

Saída esperada:

```
loss = 527.2598266601562
```

▼ 3 - Carregar a rede FaceNet

A rede FaceNet é treinada minimizando a perda de custo tripla. Mas como o treinamento exige muitos dados e muita computação, não vamos treiná-la do zero nesse trabalho. Em vez disso, carregamos um modelo treinado anteriormente.

Execute a célula abaixo para carregar a FaceNet. isso pode levar alguns minutos para ser executado.

```
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['accuracy'])  
load_weights_from_FaceNet(FRmodel)
```

Alguns exemplos de distâncias entre as codificações para três indivíduos são mostradas na Figura :



▼ 4 - Usando a rede FaceNet

▼ 4.1 - Verificação de face

Vamos construir um banco de dados contendo um vetor de codificação para cada pessoa autorizada a acessar um determinado local. Para gerar a codificação, usamos `img_to_encoding (path_da_imagem, rede)`, que basicamente executa a propagação direta da rede FaceNet para a imagem especificada.

Execute o código da célula abaixo para criar o banco de dados (representado como um dicionário python). Esse banco de dados mapeia o rosto de cada pessoa para um vetor codificado de 128 elementos e o associa ao nome da pessoa.

```
database = {}
database["joao"] = img_to_encoding("images/joao.jpg", FRmodel)
database["antonio"] = img_to_encoding("images/antonio.jpg", FRmodel)
database["miguel"] = img_to_encoding("images/miguel.jpg", FRmodel)
database["daniel"] = img_to_encoding("images/daniel.jpg", FRmodel)
database["jose"] = img_to_encoding("images/jose.jpg", FRmodel)
database["pedro"] = img_to_encoding("images/pedro.jpg", FRmodel)
database["maria"] = img_to_encoding("images/maria.jpg", FRmodel)
```

Agora, quando alguém quiser acessar o local restrito e passa o cartão de identificação que define o nome, pode-se procurar a codificação no banco de dados e usá-la para verificar se essa pessoa corresponde ao nome na identificação.

▼ Exercício #2: Implementação da função `verify()`

A função `verify()` verifica se a imagem do rosto da pessoa querendo acessar o ambiente restrito (`path_da_imagem`) é de fato da pessoa do cartão de identidade apresentado. Para isso fazer essa função, você precisa seguir as seguintes etapas:

1. Calcule a codificação da imagem definida em `image_path`;

2. Calcule a distância entre essa codificação e a codificação da imagem da identidade armazenada no banco de dados;
3. Libere o acesso se a distância for menor que 0,7, caso contrário, não libere.

Nesse cálculo você deve calcular a distância entre os dois vetores de codificação e não o quadrado dessa distância com é feito na função de custo tripla, ou seja, deve-se calcular a seguinte expressão:

$$|| f(R) - f(P) ||_2$$

Para fazer esse cálculo utilize as funções numpy `np.linalg.norm` e `np.subtract`.

PARA VOCÊ FAZER: Função `verify`

```
def verify(image_path, identity, database, model):
    """
    Função que verifica se a pessoa na "image_path" é a da "identidade".

    Argumentos:
    image_path = path para a imagem
    identity = string, nome da pessoa que se deseja verificar a identidade
    database = dicionário python que mapeia nomes de pessoas em vetores
    model = RNA inception

    Returns:
    dist -- distance between the image_path and the image of "identity" in the database.
    door_open -- True, if the door should open. False otherwise.
    """

    ### COMECE AQUI ###

    # Passo 1: Calcule o vetor de codificação da imagem, use img_to_encoding(), veja exemplo acima (≈ 1 linha)
    encoding = img_to_encoding( image_path, model )

    # Passo 2: calcula distância entre vetor de codificação da imagem e vetor de codificação da identificação (≈ 1 linha)
    dist = np.linalg.norm( np.subtract( encoding, database[identity] ) )

    # Passo 3: Permite acesso se dist < 0.7, senão bloqueia acesso (≈ 3 linhas)
    if dist < 0.7:
        print('É o {0}, seja bem vindo!'.format( identity ))
        door_open = True
    else:
```


```
print('Não é o {0}, por favor se identifique novamente!'.format( identity ))
door_open = False
### TERMINE AQUI ###

return dist, door_open
```

João está tentando entrar no ambiente restyrito e uma imagem dele é adquirida pela câmera ("images/joao02.jpg"). Execute a célula abaixo com o algoritmo de verificação de face com essa image:



```
verify("images/joao03.jpg", "joao", database, FRmodel)
```

 É o joao, seja bem vindo!
(0.6305858, True)


Saída esperada:

É o joao, seja bem vindo!

Uma pessoa desconhecida está tentando acessar o ambiente com a identificação do Pedro. A imagem dessa pessoa é adquirida pela câmera ("images/daniel.jpg"). Execute o algoritmo de verificação de face da célula abaixo para verificar se essa pessoa pode entrar.



```
verify("images/daniel02.jpg", "pedro", database, FRmodel)
```

 Não é o pedro, por favor se identifique novamente!
(1.0986296, False)

Saída esperada:

Não é o pedro, por favor se identifique novamente!

4.2 - Reconhecimento de face

Na medida em que o sistema de verificação de face está funcionando corretamente, vamos agora modificá-lo para transformá-lo em um sistema de reconhecimento de face. Em um sistema de reconhecimento de face não é necessário que os usuários utilizem a sua identificação para acessar a área restrita; uma pessoa autorizada simplesmente para em frente a porta e a porta se abre para ela.

Você vai implementar um sistema de reconhecimento de face que recebe como entrada uma imagem e decide se essa pessoa está autorizada, ou não, a acessar a área restrita. Diferentemente do sistema de verificação de face, não é necessário fornecer a identidade da pessoa como outra entrada.

▼ Exercício #3: Implementação da função `who_is_it()`.

Para implementar a função de reconhecimento de face `who_is_it()`, você tem que seguir as seguintes etapas:

1. Calcule o vetor de codificação da imagem do caminho especificado em `image_path`. Denominaremos esse vetor de vetor alvo.
2. Ache o vetor de codificação do banco de dados que tem a menor distância do vetor de codificação da imagem de entrada (vetor alvo).
 - Inicialize a variável `min_dist` com um número grande, por exemplo 100.
 - Inicialize a variável `identity` com `None`.
 - Percorra o dicionário do banco de dados, que contém os vetores de codificação e os nomes das pessoas associadas. Use o comando `for(name, db_enc) in database.items()`. Observa-se que esse comando é próprio da linguagem Python para buscar itens em um dicionário.
 - Percorrendo o banco de dados, calcule a distância entre o vetor alvo e cada um dos vetores do banco de dados (`dist`). Se a cada verificação a distância `dist` for menor do que `min_dist`, então substitua `min_dist` por `dist` e `identity` pelo nome da pessoa no banco de dados correspondente ao item sendo comparado.

```

# PARA VOCE FAZER: Função who_is_it()

def who_is_it(image_path, database, model):
    """
    Implementa reconhecimento de face identificando quem é a pessoa em image_path image.

    Argumentos:
    image_path = caminho para uma image
    database = banco de dados que contém os vetores de codificação das feces e nomes das pessoas associadas
    model = RNA FaceNet

    Retorna:
    min_dist = distância mínima entre o vetor de codificação da image_path e os vetores de codificação do banco de dados
    identity = string com o nome dapessoa em image_path
    """

    ### COMECE AQUI ###

    ## Passo 1: calcula o vetor de codificação alvo; use img_to_encoding(), veja exemplo acima ## (~ 1 linha)
    encoding = img_to_encoding( image_path, model )

    ## Passo 2: Acho a codificação de menor distância

    # Inicializa "min_dist" com um número grande, , por exemplo 100, e indentity com None (~ 2 linhas)
    min_dist = 1000
    identity = None

    # Percorra o dicionário do banco de dados com os vetores de codificação e nomes
    for (name, db_enc) in database.items():

        # Calcule a distância entre o vetor alvo e o vetor de codificação corrente "emb" do banco de dados (~ 1 linha)
        dist = np.linalg.norm( np.subtract( encoding, database[name] ) )

        # Se a distância for menor do que min_dist, então atribua dist em min_dits e o nome em identity to name. (~ 3 linhas)
        if dist < min_dist:
            min_dist = dist
            identity = name

    ### TERMINE AQUI ###

    if min_dist > 0.7:
        print("Não está na base de dados!")
    else:
        print( "É " + str(identity) + " a distância é " + str(min_dist) )

```

```

        print ( 'É ' + str(identity) + ' , a distancia é ' + str(min_dist))

    return min_dist, identity

```

Younes is at the front-door and the camera takes a picture of him ("images/camera_0.jpg"). Let's see if your who_it_is() algorithm identifies Younes.

```
who_is_it("images/joao03.jpg", database, FRmodel)
```

 É pedro, a distância é 0.57696646
(0.57696646, 'pedro')

Saída esperada:

É joao, a distância é 0.6593835

(0.6593835, 'joao')

▼ Exercício #4: teste do sistema de reconhecimento de faces

Teste o seu programa usando outras imagens, simplesmente alterando a imagem de entrada, como por exemplo, use qualquer imagem com nome camera+número.jpg e veja os resultados. Teste pelo menos 5 imagens.


```

# PARA VOCÊ FAZER: teste do sistema de reconhecimento de face

### COMECE AQUI ###
# Teste o seu sistema de reconhecimento de face (≈ 6 linhas).
# Use as imagens camera1.jpg, camera2.jpg, camera3.jpg, camera4.jpg, camera5.jpg, camera6.jpg
who_is_it("images/camera1.jpg", database, FRmodel)
who_is_it("images/camera2.jpg", database, FRmodel)
who_is_it("images/camera3.jpg", database, FRmodel)
who_is_it("images/camera4.jpg", database, FRmodel)
who_is_it("images/camera5.jpg", database, FRmodel)

```

```
who_is_it("images/camera6.jpg", database, FRmodel)
### TERMINE AQUI ###
```

 É antonio, a distância é 0.25238556
É maria, a distância é 0.6854024
É joao, a distância é 0.5926709
Não está na base de dados!
Não está na base de dados!
Não está na base de dados!
(0.85765874, 'jose')

Saída esperada:

```
É antonio, a distância é 0.25238562
É maria, a distância é 0.6854024
É joao, a distância é 0.59267086
Não está na base de dados!
Não está na base de dados!
Não está na base de dados!
```

Observa-se que essa saída é somente um exemplo. Usando outras imagens o resultado será diferente.

Referências:

- Florian Schroff, Dmitry Kalenichenko, James Philbin (2015). [FaceNet: A Unified Embedding for Face Recognition and Clustering](#)
- Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf (2014). [DeepFace: Closing the gap to human-level performance in face verification](#)
- A RNA pré-treinada usada nesse trabalho foi desenvolvida por Victor Sy Wang's e obtida de: <https://github.com/iwantooxxoxx/Keras-OpenFace>.
- Repositório oficial da FaceNet: <https://github.com/davidsandberg/facenet>

