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 dversoss arquivos:

1) Diretório com imagens

https://drive.google.com/drive/folders/1B7sIDnrH0ZpiWvNzmjUTnwblfrOlM-7Z?usp=sharing

2) Diretório com arquivos de parâmetros da rede

https://drive.google.com/drive/folders/11bAqlL2QEiYepSFIZt70b9g2YWFram3h?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 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()
```

inception_5a_1x1_bn (BatchNorma	(None, 256, 3, 3)	1024	inception_5a_1x1_conv[0][0]
activation_30 (Activation)	(None, 384, 3, 3)	0	inception_5a_3x3_bn2[0][0]
zero_padding2d_20 (ZeroPadding2	(None, 96, 3, 3)	0	activation_31[0][0]
activation_32 (Activation)	(None, 256, 3, 3)	0	inception_5a_1x1_bn[0][0]
concatenate_5 (Concatenate)	(None, 736, 3, 3)	0	activation_30[0][0] zero_padding2d_20[0][0] activation_32[0][0]
inception_5b_3x3_conv1 (Conv2D)	(None, 96, 3, 3)	70752	concatenate_5[0][0]
inception_5b_3x3_bn1 (BatchNorm	(None, 96, 3, 3)	384	inception_5b_3x3_conv1[0][0]
activation_33 (Activation)	(None, 96, 3, 3)	0	inception_5b_3x3_bn1[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 736, 1, 1)	0	concatenate_5[0][0]
zero_padding2d_21 (ZeroPadding2	(None, 96, 5, 5)	0	activation_33[0][0]
inception_5b_pool_conv (Conv2D)	(None, 96, 1, 1)	70752	max_pooling2d_5[0][0]
inception_5b_3x3_conv2 (Conv2D)	(None, 384, 3, 3)	332160	zero_padding2d_21[0][0]
inception_5b_pool_bn (BatchNorm	(None, 96, 1, 1)	384	inception_5b_pool_conv[0][0]
inception_5b_1x1_conv (Conv2D)	(None, 256, 3, 3)	188672	concatenate_5[0][0]
inception_5b_3x3_bn2 (BatchNorm	(None, 384, 3, 3)	1536	inception_5b_3x3_conv2[0][0]
activation_35 (Activation)	(None, 96, 1, 1)	0	inception_5b_pool_bn[0][0]
inception_5b_1x1_bn (BatchNorma	(None, 256, 3, 3)	1024	inception_5b_1x1_conv[0][0]
activation_34 (Activation)	(None, 384, 3, 3)	0	inception_5b_3x3_bn2[0][0]
zero_padding2d_22 (ZeroPadding2	(None, 96, 3, 3)	0	activation_35[0][0]
activation_36 (Activation)	(None, 256, 3, 3)	0	inception_5b_1x1_bn[0][0]

```
__________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 736, 3, 3)    0           activation_34[0][0]
                                                                 zero_padding2d_22[0][0]
                                                                 activation_36[0][0]
__________________________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 736, 1, 1)    0           concatenate_6[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 736)          0           average_pooling2d_3[0][0]
__________________________________________________________________________________________________
dense_layer (Dense)             (None, 128)          94336       flatten[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 128)          0           dense_layer[0][0]
==================================================================================================
Total params: 3,743,280
Trainable params: 3,733,968
Non-trainable params: 9,312
__________________________________________________________________________________________________
```

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 vetro 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 limar, pode-se determinar se as duas imagens representam a mesma pessoa.

Portanto, uma codificação é boa se:

- As codificações de duas imagens de um amesma 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 treinamemnto 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 (triplas) 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 fução de custo tripl:

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

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 imagm 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 funçaõ 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 $|| \ f(A^{(i)}) - f(N^{(i)}) \ ||_2^2$, equanto 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 distência 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 distência 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 ) )
    ### 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 :

Figure 5 (Andrew Ng, deeplearning.ai)

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)
```