EigenVargas

Lucas Geraldo dos Santos Braga & *João Pedro Giordani Donasolo*
22/11/2020

Objetivo:

Reconhecer a face de professores da EMAp com base em fotos achadas em buscas na Web. Para isso, utilizaremos Eigenfaces. A seguir, o passo-a-passo seguido no script:

Passo a passo:

0. Selecionar nosso dataset, preenchido com imagens de professores, e completo com imagens de outro dataset encontrado na web.
1. Preparar o dataset, recortando as faces dos professores conforme necessário e colocando fundo branco.
2. Dessas imagens, pegar uma para treinamento e deixar as outras como teste (feita renomeando as de teste com números no final).
3. Recortar as imagens em um tamanho padronizado e convertê-las para preto e branco 8bits(feito em outro notebook).
4. Achatar as imagens, de matrizes para vetores
5. Calcular a face média das imagens de treinamento.
6. Normalizar as imagens de treinamento, de cada uma delas, subtrair a face média
7. Calcular a matriz de convariância,
8. Extrair os autovetores,
9. Calcular as Eigenfaces - Autovetores x Faces normalizadas
10. Calcular os pesos das imagens.

Fundamentação teórica

A aceitação ou recusa de uma imagem como face ou não se dá pela mensuração da sua diferença da imagem em contraste ao que o programa considera como face, isto é, o erro da projeção da imagem de entrada sobre o espaço vetorial gerado pelas eigenfaces.

O procedimento é feito colapsando as informações das imagens (pixels) em vetores unidimensionais, que irão compor as linhas da matriz. Com a matriz em mãos, podemos realizar sua decomposição SVD para descobrir seus autovalores e autovetores. Os autovetores, presentes na matriz U, terão o número de pixels de uma imagem, e, após o redimensionamento, podem ser visualizados como faces, isto é, "eigenfaces" (fazendo uma alusão ao termo "eigenvectors", ou autovetores, em inglês).

Por fim, o reconhecimento ou não de uma imagem de entrada como face se dará pelo erro da projeção no espaço gerado pelas eigenfaces. No caso deste ser numericamente maior que um limite imposto, será recusado. Caso seja menor, será aceito. De forma complementar, o nosso programa tenta identificar a qual imagem de treinamento o input se relaciona.

Resultados

Tivemos duas abordagens similares impostas pela natureza do dataset. Na primeira abordagem, tentantos puramente reconhecer faces, e, se possível atrelá-las a um individuo, usando uma maçã como objeto de controle. Obtivemos taxa de sucesso de **11/18**. Na segunda, tentamos dar match dos inputs com os professores corretos, obtendo assim uma taxa de sucesso de **8/12**

Importação os pacotes necessários

```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
import os
import re
```

Definindo o diretório das imagens, listando as mesmas e definindo a altura e largura das imagens.

```
dataset_path = "imgs/"
images  = os.listdir(dataset_path)
print(images)
w  = 195
h = 231
```

```
['bteste.jpg', 'dteste_1.jpg', 'Wagner.jpg', 'Yuri.jpg', 'Renato_1.jpg',
'Camacho_1.jpg', 'cteste_1.jpg', 'cteste.jpg', 'Yuri_1.jpg', 'Wagner_1.jpg',
'apple1_gray.jpg', 'dteste.jpg', 'Renato_7.jpg', 'ateste_1.jpg',
'Yuri_2.jpg', 'Wagner_2.jpg', 'Camacho_4.jpg', 'Renato_3.jpg',
'Camacho_3.jpg', 'Renato.jpg', 'Renato_2.jpg', 'Camacho_2.jpg',
'bteste_1.jpg', 'Camacho.jpg', 'Wagner_3.jpg', 'ateste.jpg']
```

Aqui foram definidas as imagens de teste e de treinamento.

- train_images - Imagens de treinamento, usadas para identificar as outras, basicamente as que não possuem números no nome;
- test_images - Imagens que serão testadas no final.

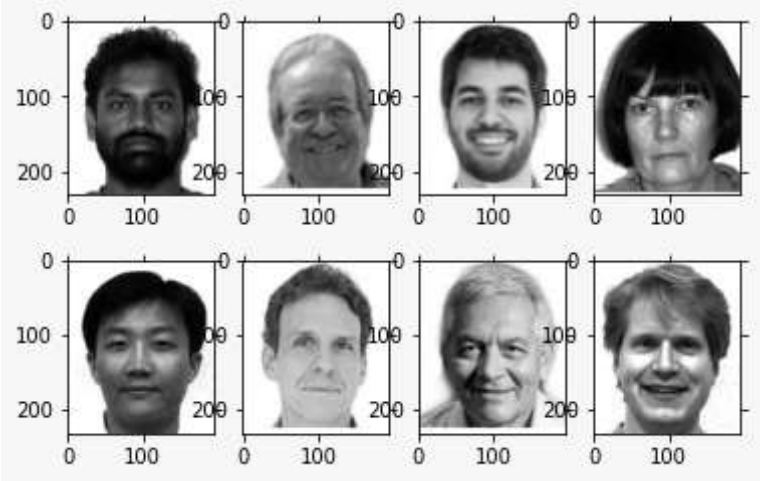
```
train_images = []
test_images = []
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

for i in images:
    test = hasNumbers(i)
    if test == False:
        train_images.append(i)
    else:
        test_images.append(i)

train_images
```

```
['bteste.jpg',
 'Wagner.jpg',
 'Yuri.jpg',
 'cteste.jpg',
 'dteste.jpg',
 'Renato.jpg',
 'Camacho.jpg',
 'ateste.jpg']
```

Criação do array que irá conter as imagens de treinamento

training_tensor é criada como um ndarray de tamanho fixo (quantidade de imagens por (altura x largura)) e contém valores em formato float64.
A estrutura de repetição pegará cada imagem definida como de treinamento e adicionará ao Array.

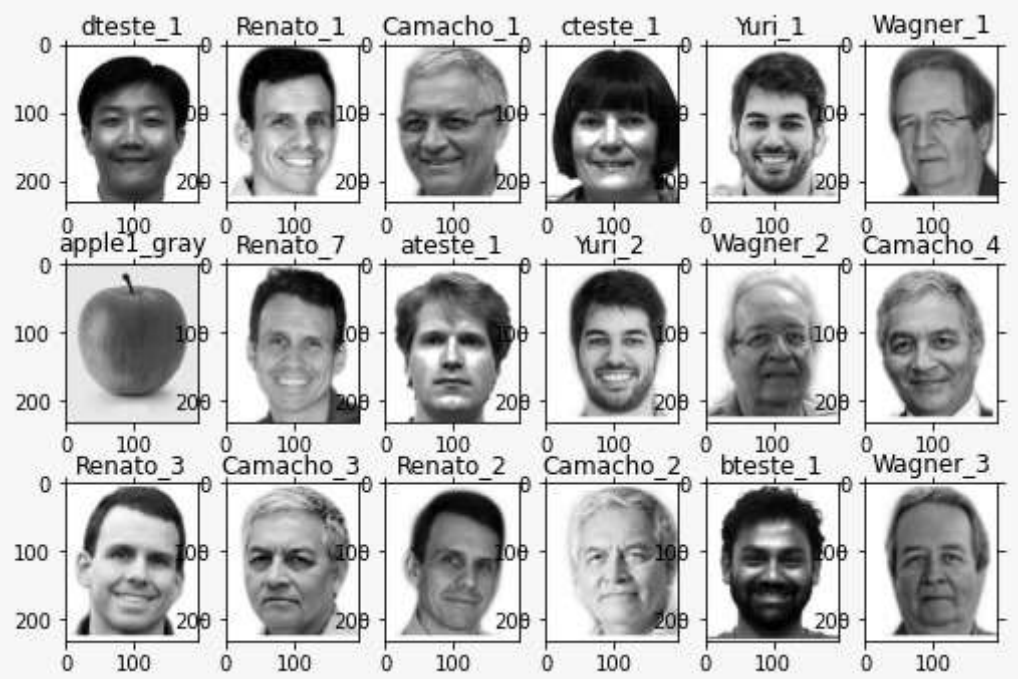
```
training_tensor = np.ndarray(shape=(len(train_images), h*w),
dtype=np.float64)

for i in range(0, len(train_images)):
    img = plt.imread(dataset_path + train_images[i])
    training_tensor[i,:] = np.array(img, dtype='float64').flatten()
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='gray')

plt.show()

```



O mesmo do passo anterior será feito com as imagens de teste.

```
print('Test Images:')
testing_tensor = np.ndarray(shape=(len(test_images), h*w), dtype=np.float64)
# Essa é a matriz com as imagens que serão testadas posteriormente

for i in range(len(test_images)):
    img = imread(dataset_path + test_images[i])
    testing_tensor[i,:] = np.array(img, dtype='float64').flatten()
    plt.subplot(3,6,1+i)
    plt.title(test_images[i].split('.')[0])
    plt.imshow(img, cmap='gray')
    plt.subplots_adjust(right=1.2, top=1.2)

plt.show()
```

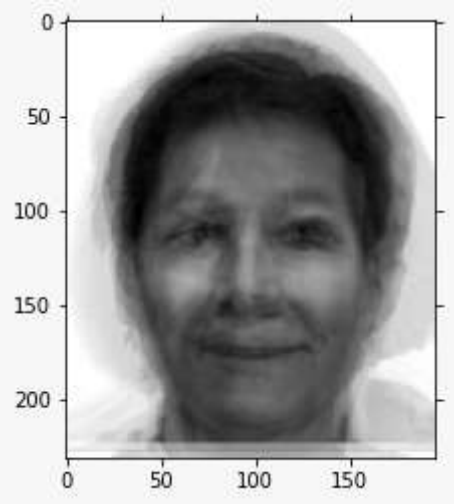
```
Test Images:
```



Face média

Nesse passo é feito o calculo da face média, recursivamente somando cada imagem do training_tensor ao array de zeros criado para a mean_face, e dividindo pelo total de faces ao final

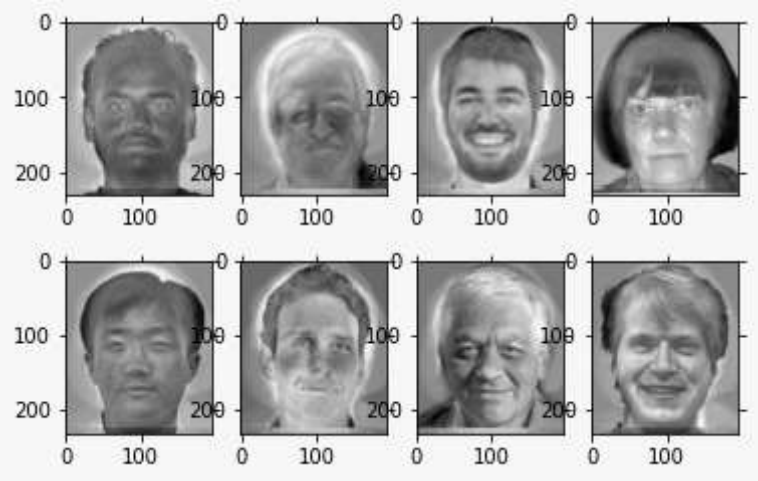
```
mean_face = np.zeros((1,h*w))

for i in training_tensor:
    mean_face = np.add(mean_face,i)

mean_face = np.divide(mean_face,float(len(train_images))).flatten()

plt.imshow(mean_face.reshape(h, w), cmap='gray')

plt.show()
```



Faces normalizadas

Aqui as faces são normalizadas, subtraindo de cada uma a face média.

```
normalised_training_tensor = np.ndarray(shape=(len(train_images), h*w))

for i in range(len(train_images)):
    normalised_training_tensor[i] =
np.subtract(training_tensor[i],mean_face)

for i in range(len(train_images)):
    img = normalised_training_tensor[i].reshape(h,w)
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='gray')

plt.show()
```



Matriz de Covariância

Aqui, é calculada a matriz covariância das faces normalizadas, e, então, extraidos desta os autovalores e autovetores. Por conveniência, os mesmos são agrupados em pares e ordenados em ordem decrescente.
Matriz AA^t

```
cov_matrix = np.cov(normalised_training_tensor)
cov_matrix = np.divide(cov_matrix,8.0)

eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix) # Autovalores e autovetores de AA^t, ou seja, matriz V

eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]

# Ordenando:
eig_pairs.sort(reverse=True)
eigvalues_sort  = [eig_pairs[index][0] for index in range(len(eigenvalues))]
eigvectors_sort = [eig_pairs[index][1] for index in range(len(eigenvalues))]
```

Variância Cumulativa

Análise PCA dos componentes do treinamento, por meio do somatório da variância dos autovalores (já ordenados).

```
var_comp_sum = np.cumsum(eigvalues_sort)/sum(eigvalues_sort)

print(f"Proporção cumulativa de variância em relação aos componentes: \n{var_comp_sum}")

num_comp = range(1,len(eigvalues_sort)+1)
plt.title('Proporção cumulativa da variancia em relação aos componentes')
plt.xlabel('componentes principais')
plt.ylabel('Proporção da variancia cumulativa')

plt.scatter(num_comp, var_comp_sum)
plt.show()
```

```
Proporção cumulativa de variância em relação aos componentes:
[0.33338766 0.57810348 0.71562045 0.81654941 0.91096097 0.96489788
 1.         1.        ]
```


Proporção cumulativa da variancia em relação aos componentes
Proporção da variancia cumulativa
componentes principais
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8

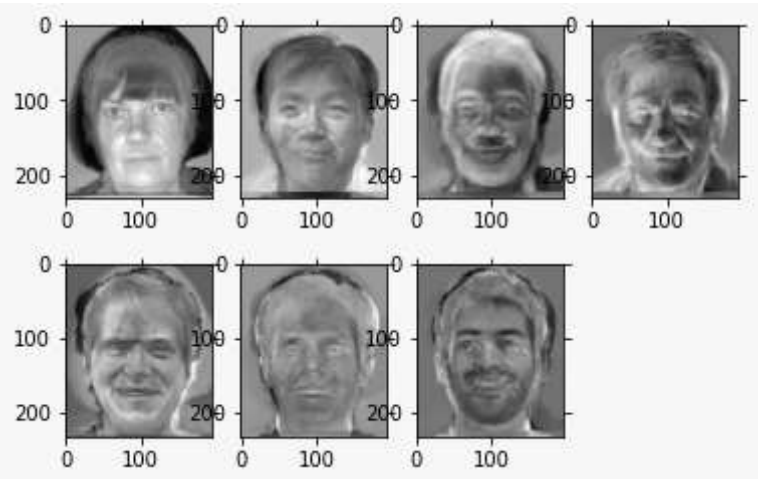
Matriz V^t

```
reduced_data = np.array(eigvectors_sort[:7]).transpose() # Matriz V^t
reduced_data
```

```
array([[-0.35099908,  0.39901424, -0.00586678,  0.12124807, -0.45152026,
         0.47611397, -0.38379254],
       [-0.287197  , -0.51327757,  0.5856991 ,  0.38290987, -0.08602526,
        -0.09261954,  0.15304213],
       [-0.06340159, -0.06902412, -0.62606537,  0.33413226,  0.0911486 ,
         0.19541926,  0.56224277],
       [ 0.86992091, -0.01853928,  0.13100204,  0.05903928, -0.29580439,
         0.0987027 , -0.00206784],
       [-0.11698869,  0.51841604,  0.0937055 , -0.1786501 , -0.17037371,
        -0.6598834 ,  0.29561558],
       [ 0.03744889, -0.13931554, -0.32521257,  0.24315438,  0.31343611,
        -0.41073468, -0.64988947],
       [-0.12955438, -0.45493509, -0.17411105, -0.77241896, -0.13984361,
         0.06339945, -0.02698687],
       [ 0.04077094,  0.2776613 ,  0.32084914, -0.1894148 ,  0.73898253,
         0.32960223,  0.05183624]])
```

```
eigenfaces = np.dot(training_tensor.transpose(),reduced_data) # Sendo u = Av^t
eigenfaces = eigenfaces.transpose() # Eigenfaces
```

```
for i in range(eigenfaces.shape[0]):
    img = eigenfaces[i].reshape(h,w)
    plt.subplot(2,4,1+i)
    plt.imshow(img, cmap='gray')

plt.show()
```



Reconhecimento facial

Aqui, seguiremos basicamente 3 passsos para o reconhecimento:

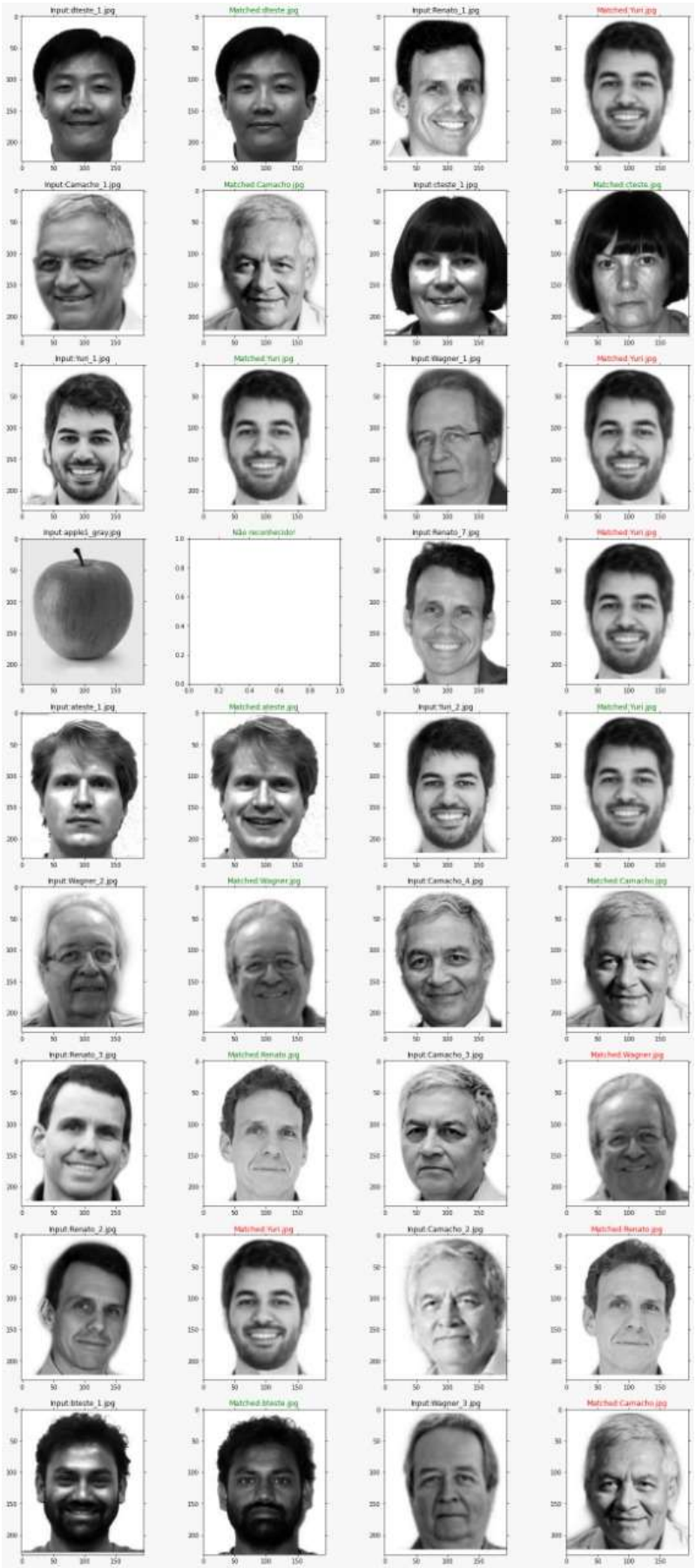
- Vetorizar e normalizar cada imagem: subtraindo a média calculada pela mesma.
- Calcular os pesos: multiplicar as eigenfaces pela média calculada.
- Interpretar a diferença do vetor peso pelo peso da imagem, se é grande, não é uma face, se é pequena, é uma face (usamos o peso da maçã como parâmetro)
- Após isso, usamos um segundo parâmetro, para descobrir se a face reconhecida pertence a um dos participantes.

```
weights = np.array([np.dot(eigenfaces,i) for i in
normalised_training_tensor])
weights
```

```
array([[-4.33701228e+07,  1.75629373e+08,  3.33928865e+07,
         4.70076150e+06, -1.13156324e+08,  5.90386547e+07,
        -9.67476261e+06],
       [-1.45265003e+08, -1.82687875e+08,  5.21652551e+07,
         4.27680640e+07,  4.63435434e+07, -3.06045661e+07,
         2.66598155e+06],
       [-6.13644033e+07, -6.38920448e+07, -1.03311915e+08,
         3.68074411e+07,  5.04578797e+07, -9.83630406e+06,
         1.71256416e+07],
       [ 3.93751381e+08,  1.27139427e+08,  7.28073478e+07,
        -5.80290094e+06, -1.43559478e+08,  6.05639849e+07,
         5.37228297e+06],
       [-3.29908494e+05,  1.66201751e+08,  3.07684463e+07,
        -2.12667539e+07, -5.40767817e+07, -1.61196448e+07,
         1.18375005e+07],
       [-6.55328504e+07, -1.23180313e+08, -8.12429379e+07,
         3.16748520e+07,  1.08548574e+08, -5.94331226e+07,
        -2.58019247e+07],
       [-8.58919232e+07, -1.58963997e+08, -4.47459187e+07,
        -7.09126439e+07,  3.26458164e+07, -1.82876601e+07,
        -3.03750976e+06],
       [ 8.00283021e+06,  5.97536774e+07,  4.01668358e+07,
        -1.79688198e+07,  7.27967711e+07,  1.46786580e+07,
         1.51279038e+06]])
```

```
count, num_imgs, acertos = 0, 0, 0
def eigentest(img, train_images,eigenfaces,weights):
    global count,highest_min,num_imgs,acertos
    face_test = plt.imread(dataset_path+img)
    num_imgs += 1
    face_test_v= np.array(face_test, dtype='float64').flatten()
    normalised_face_test = np.subtract(face_test_v,mean_face)

    plt.subplot(9,4,1+count)
    plt.imshow(face_test, cmap='gray')
    plt.title("Input:"+'.'.join(img.split('.')[:2]))

    count+=1

    weights_test = np.dot(eigenfaces, normalised_face_test)
    dif  = weights - weights_test
    normas = np.linalg.norm(dif, axis=1)
    index = np.argmin(normas)

    t1 = 165691852
    t0 = 145000000

    if normas[index] < t1:
        plt.subplot(9,4,1+count)
        if normas[index] < t0:
            if img.split('_')[0] == train_images[index].split('.')[0]:
                plt.title("Matched:"+'.'.join(train_images[index].split('.')
[:2]), color='g')
                plt.imshow(imread(dataset_path + train_images[index]),
cmap="gray")
                acertos += 1
            else:
                plt.title("Matched:"+'.'.join(train_images[index].split('.')
[:2]), color='r')
                plt.imshow(imread(dataset_path + train_images[index]),
cmap='gray')
        else:
            if img.split('_')[0] not in [i.split('.')[0] for i in
train_images]:
                plt.title("Não reconhecido!", color="g")
                acertos += 1
            else:
                plt.title("Não reconhecido", color="r")

        plt.subplots_adjust(right=1.2, top=2.5)
    else:
        plt.subplot(9,4,1+count)
        if img.split('_')[0] != "apple1":
            plt.title("Não reconhecido!", color="r")
        else:
            plt.title("Não é uma face!", color="g")
            acertos += 1

    count+=1

fig = plt.figure(figsize=(15, 15))
```

```
for i in range(len(test_images)):
    eigentest(test_images[i], train_images,eigenfaces,weights)

plt.show()

print(f"Previsões corretas: {acertos}/{num_imgs} =
{acertos/num_imgs*100.00}%")

```

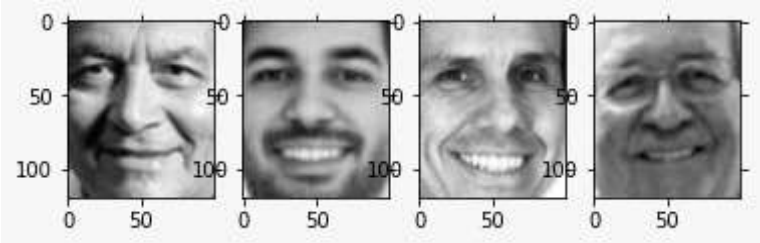



```
Previsões corretas: 11/18 = 61.11%
```

Após isso, resolvemos testar o reconhecimento pessoal em si

Para isso, cortamos a face de modo a centralizar no rosto, e manter somente detalhes que individualizem o objeto de estudo, não faria sentido testar se é humano ou não, há a pura de atrelar uma imagem ao seu indivíduo. Pouparemos o código, porém, o mesmo foi somente com os professores:

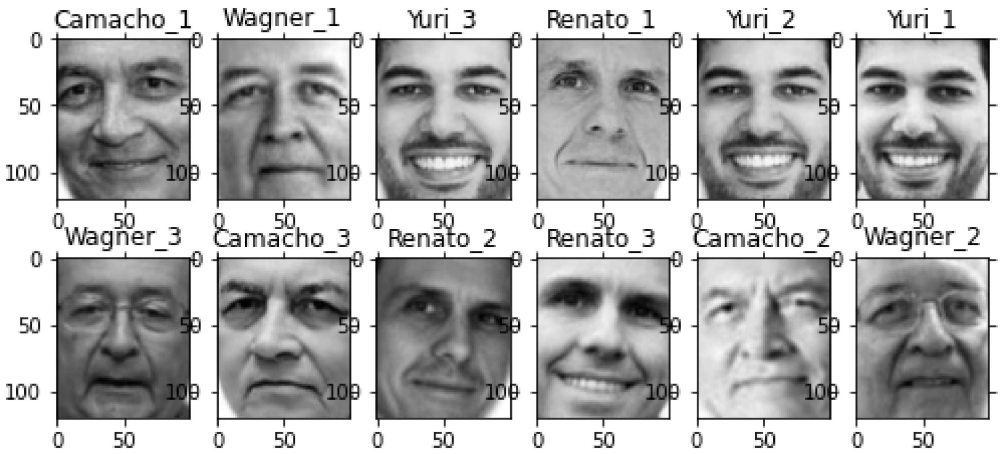
Imagens de treinamento



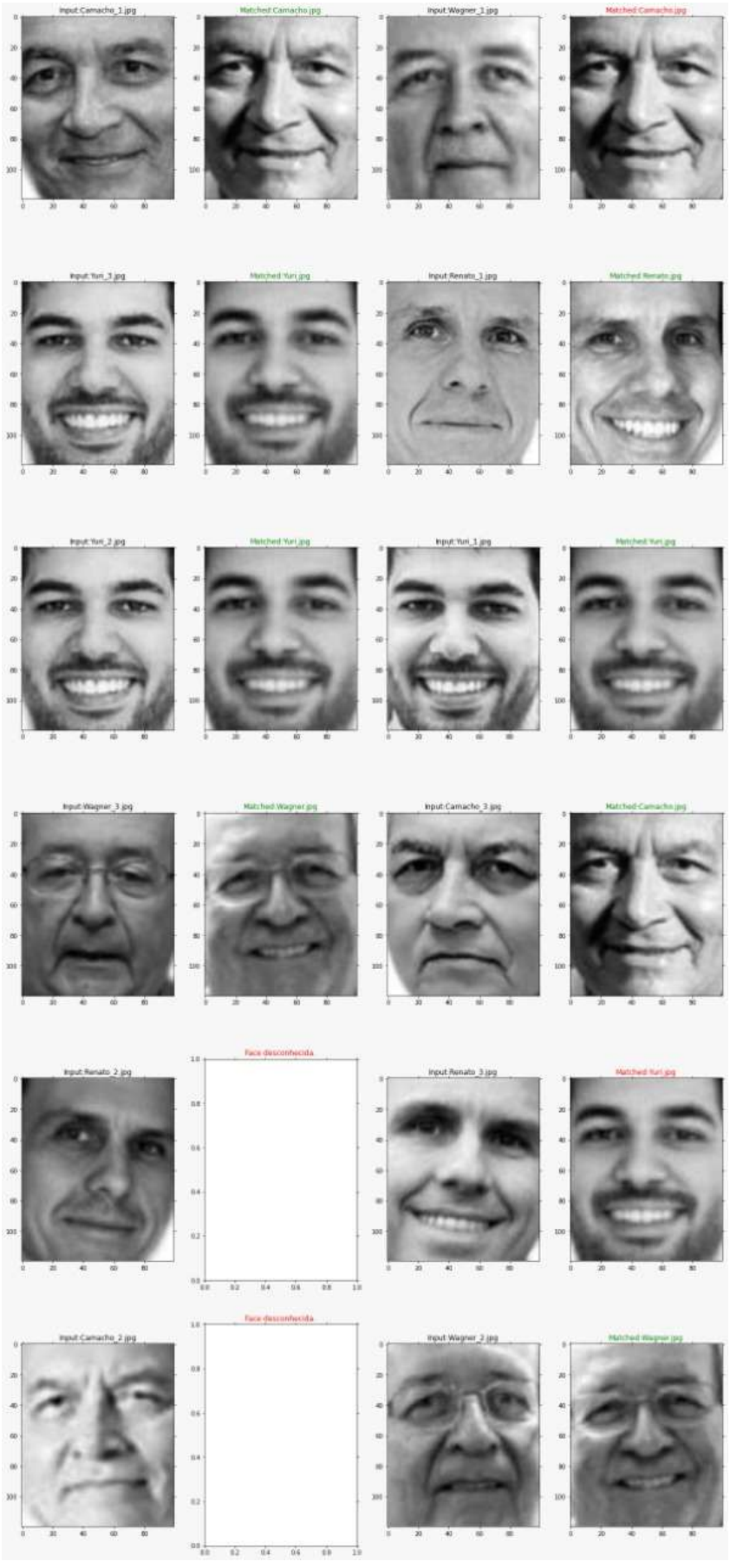

```
count, num_imgs, acertos = 0, 0, 0
def eigentest(img, train_images,eigenfaces,weights):
    global count,highest_min,num_imgs,acertos
    face_test = plt.imread(dataset_path+img)
    num_imgs += 1
    face_test_v= np.array(face_test, dtype='float64').flatten()
    normalised_face_test = np.subtract(face_test_v,mean_face)

    plt.subplot(6,4,1+count)
    plt.imshow(face_test, cmap='gray')
    plt.title("Input:"+'.'.join(img.split('.')[:2]))

    count+=1

    weights_test = np.dot(eigenfaces, normalised_face_test)
    dif  = weights - weights_test
    normas = np.linalg.norm(dif, axis=1)
    index = np.argmin(normas)
    t0 = 41800000

    plt.subplot(6,4,1+count)
    if normas[index] < t0:
        if train_images[index].split(".")[0] in img.split("_")[0]:
            plt.title("Matched:"+'.'.join(train_images[index].split('.')
[:2]), color='g')
            plt.imshow(imread(dataset_path+train_images[index]),
cmap='gray')
            acertos += 1
        else:
            plt.title("Matched:"+'.'.join(train_images[index].split('.')
[:2]), color='r')
            plt.imshow(imread(dataset_path+train_images[index]),
cmap='gray')
    else:
        if img.split('_')[0] not in [i.split('.')[0] for i in train_images]:
            plt.title('Face desconhecida', color='g')
            acertos += 1
```

```
        else:
            plt.title("Face desconhecida.", color='r')

    plt.subplots_adjust(right=1.2, top=2.5)
    count+=1

fig = plt.figure(figsize=(15, 15))
for i in range(len(test_images)):
    eigentest(test_images[i], train_images,eigenfaces,weights)

plt.show()

print(f"Correct predictions: {acertos}/{num_imgs} =
{round(acertos/num_imgs*100, 2)}")
```

Imagens testadas



Resultados



☑ **Previsões corretas: 8/12 = 66.67%**

Conclusão

Ao longo do trabalho, foi possível ver que, com os conhecimentos adquiridos até aqui na disciplina, é possível identificar semelhanças em imagens com certo grau de confiança. Ao tentar diferenciar pessoas de objetos, perdíamos a precisão na identificação de indivíduos, muito provavelmente pela quantidade de branco (valor 255) ao fundo, distorcendo assim as métricas utilizadas e poluindo o que realmente importa na identificação: as peculiaridades de cada indivíduo. Recortando e aproximando na face, perdemos consideravelmente a habilidade de distinguir objetos (diferenciação feita com t0), porém, aumentamos a capacidade de distinguir indivíduos (via t1), visto que, as novas matrizes formadas, representam 100% do objeto de estudo: A face de cada um.

Referências

[1.] EigenFaces and A Simple Face Detector with PCA/SVD in Python **- SANDIPANWEB**

[2.] Eigenfaces — Face Classification in Python **- towardsdatascience**

[3.] Restore from **"roger-" - Github**

[4.] Explanation on face recognition using Eigenfaces **- Lipman's Artificial Intelligence Directory**

[5.] Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control - **Steven L. Brunton & J. Nathan Kutz**

[6.] Eigenface from **"pyofey" - Github**