aplicacao

November 14, 2019

[1]:
```
import numpy as np
import pandas as pd
import face_recognition
import cv2
import matplotlib.pyplot as plt
import dlib
from imutils import face_utils
from imutils import paths
import os
import pickle
```

1 Cascade Classifiers

Esse algoritmo faz os seguintes passos:

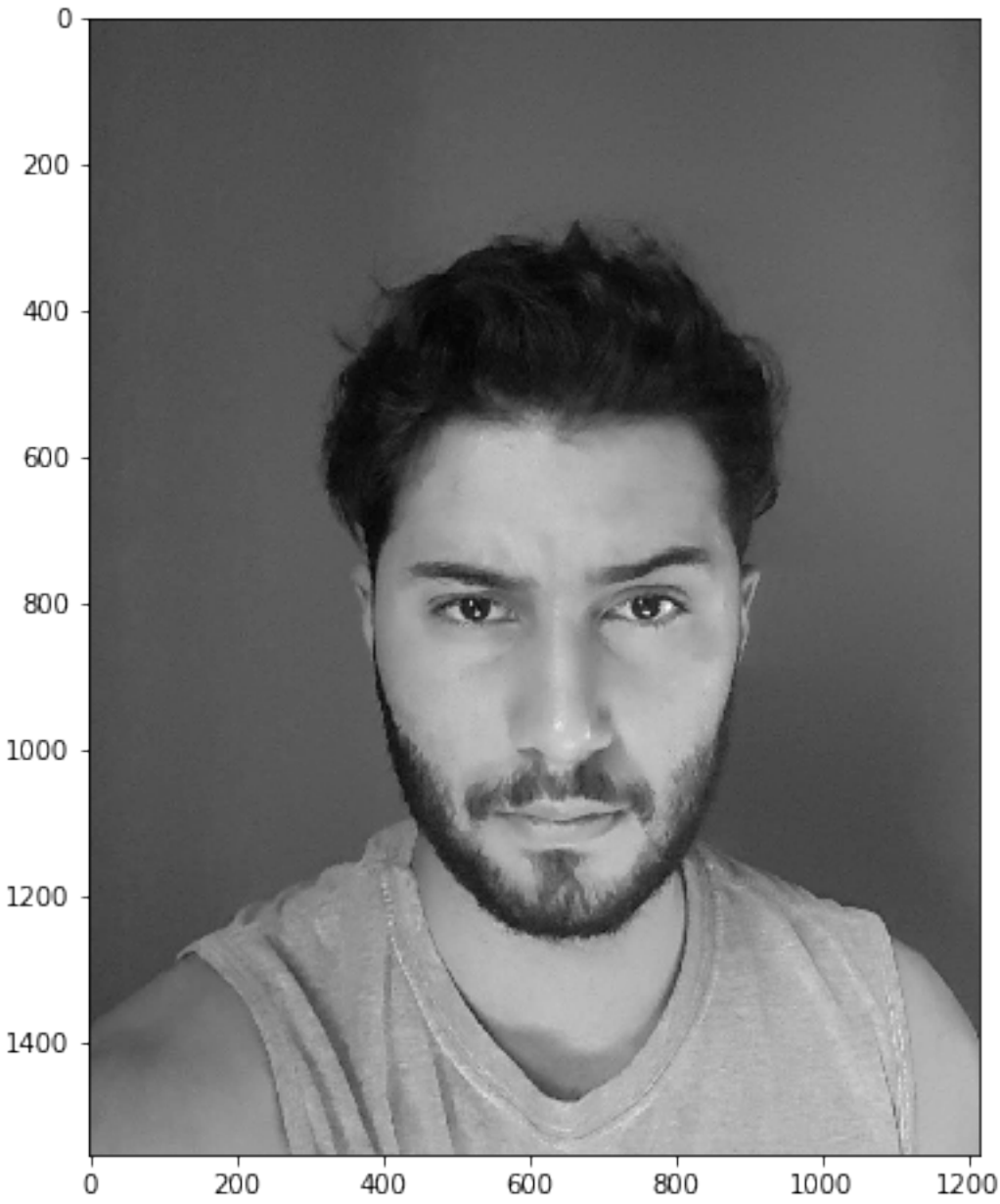
- Extração das Haar Feature (retângulos)
- Cria uma imagem integral
- Treinamento com Adaboost
- Cascading Classifiers

Importando os modelos treinados

[17]:
```
cascPath = "C:/Users/Henrique/Anaconda3/Lib/site-packages/cv2/data/
 ↪haarcascade_frontalface_default.xml"
eyePath = "C:/Users/Henrique/Anaconda3/Lib/site-packages/cv2/data/
 ↪haarcascade_eye.xml"
smilePath = "C:/Users/Henrique/Anaconda3/Lib/site-packages/cv2/data/
 ↪haarcascade_smile.xml"

faceCascade = cv2.CascadeClassifier(cascPath)
eyeCascade = cv2.CascadeClassifier(eyePath)
smileCascade = cv2.CascadeClassifier(smilePath)
```

Carregando imagem

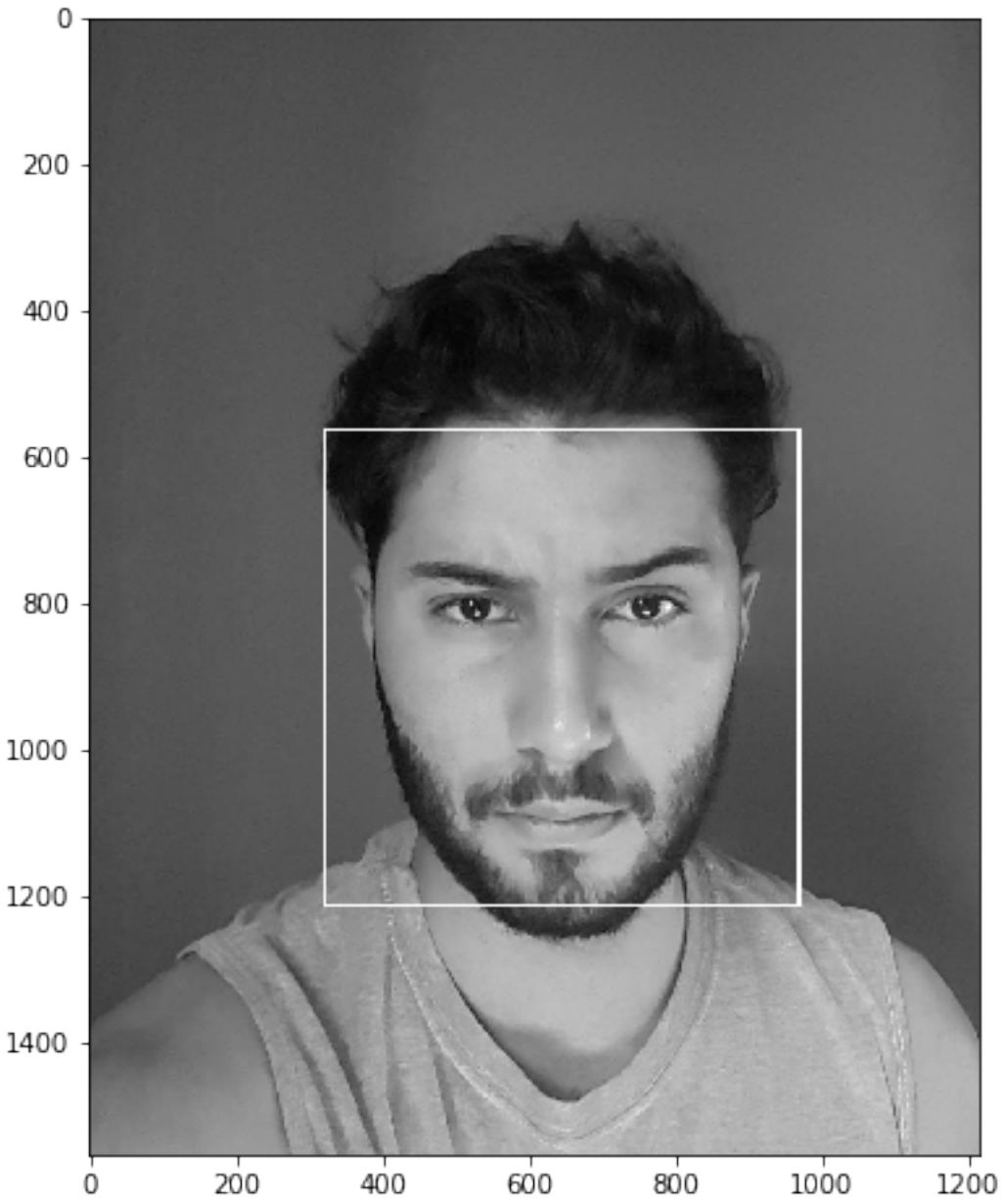
[18]:
```
gray = cv2.imread('selfie2.jpg', 0)

plt.figure(figsize=(12,8))
plt.imshow(gray, cmap='gray')
plt.show()
```



Utilizando o detector na imagem

[19]:
```
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    flags=cv2.CASCADE_SCALE_IMAGE
)


for (x, y, w, h) in faces:
    cv2.rectangle(gray, (x, y), (x+w, y+h), (255, 255, 255), 3)
```

[20]:
```
plt.figure(figsize=(12,8))
plt.imshow(gray, cmap='gray')
plt.show()
```



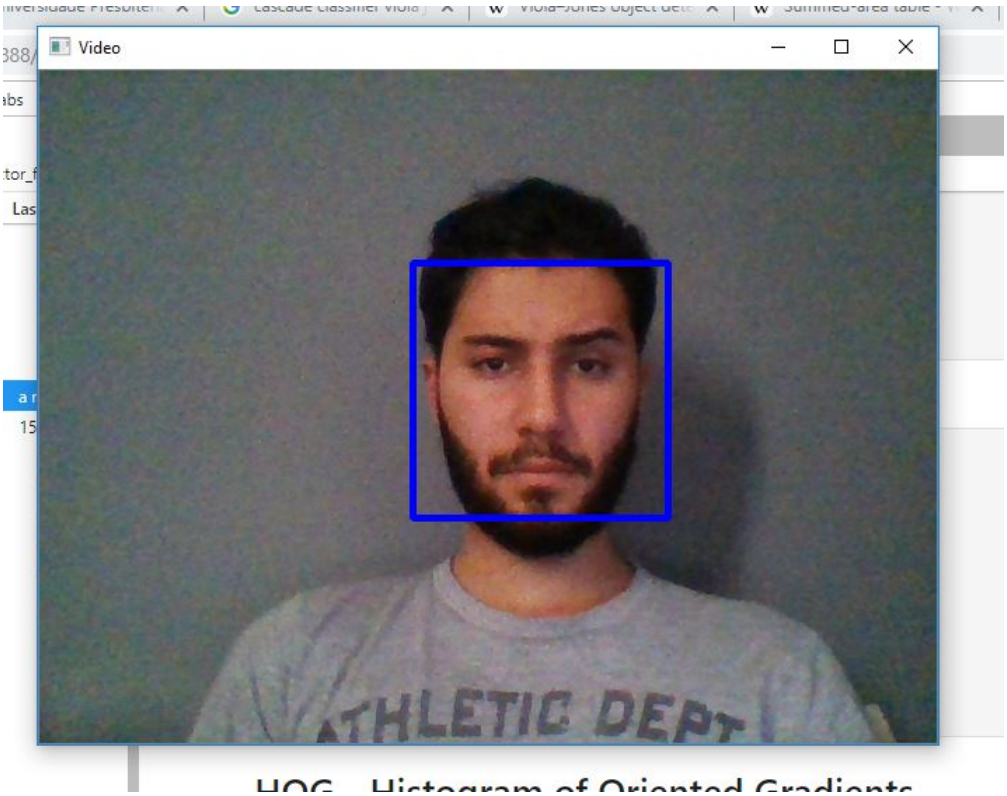
Função para detecção de faces - Cascade

[21]:
```
def detect_faces_cascade(frame):

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
```

```
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    for (x, y, w, h) in faces:
        if w > 100 :
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 3)
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = frame[y:y+h, x:x+w]
```

Utilizando a web-cam como input

[23]:
```
video_capture = cv2.VideoCapture(0)

while True:
    ret, frame = video_capture.read()

    detect_faces_cascade(frame)

    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```



title

2 HOG - Histogram of Oriented Gradients

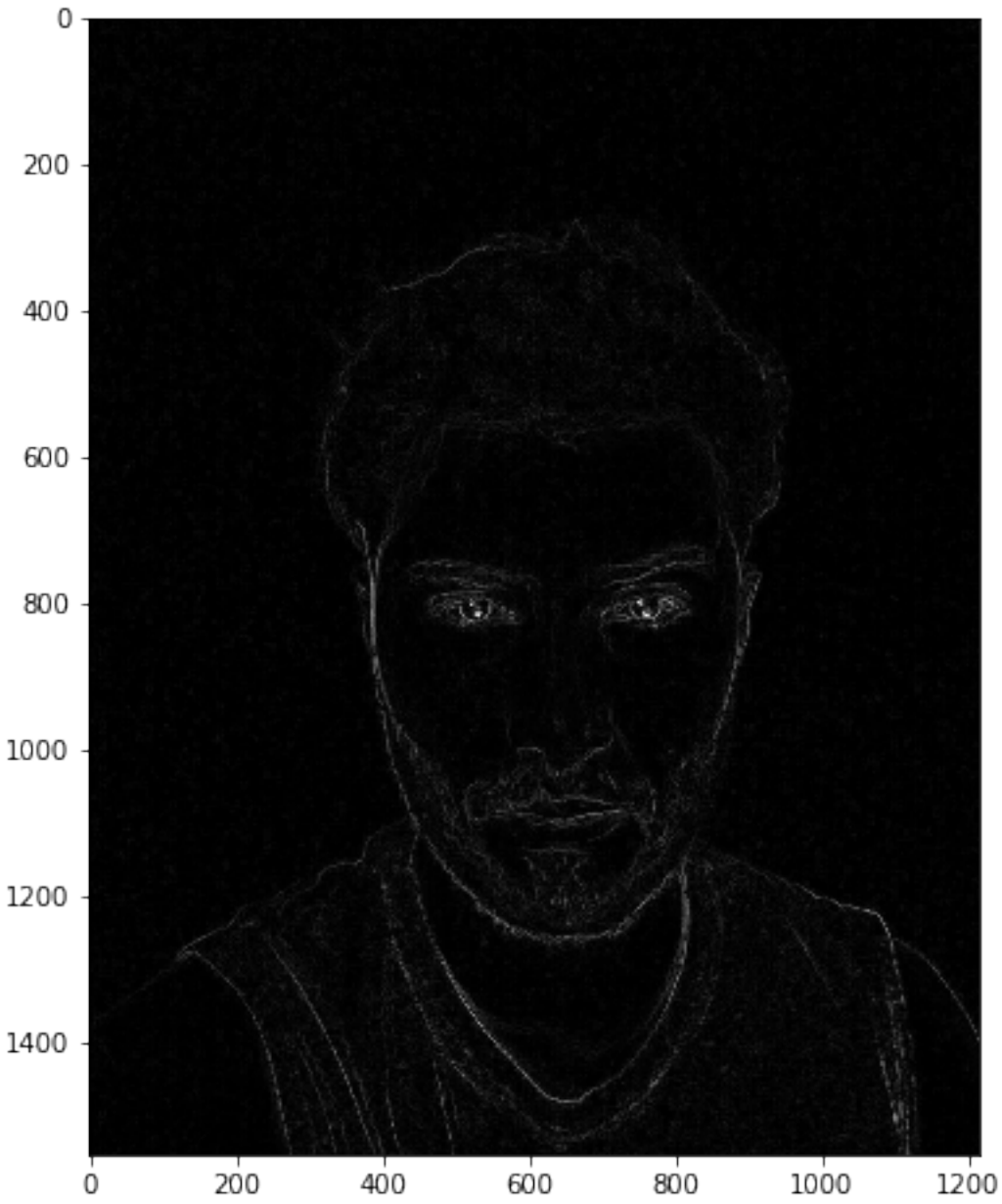
Esse algoritmo faz os seguintes passos:

- Calcular os gradientes de cada pixel
- Utilizar uma máscara 16x16 para determinar majoritariamente a direção do gradiente da região
- Treinamento dos padrões de rosto
- Identificar os rostos de acordo com os padrões treinados

Carregando a imagem e calculando os gradientes

[24]:
```
gray = cv2.imread('selfie2.jpg', 0)

im = np.float32(gray) / 255.0

#calculando os gradientes
gx = cv2.Sobel(im, cv2.CV_32F, 1, 0, ksize=1)
gy = cv2.Sobel(im, cv2.CV_32F, 0, 1, ksize=1)
mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
```

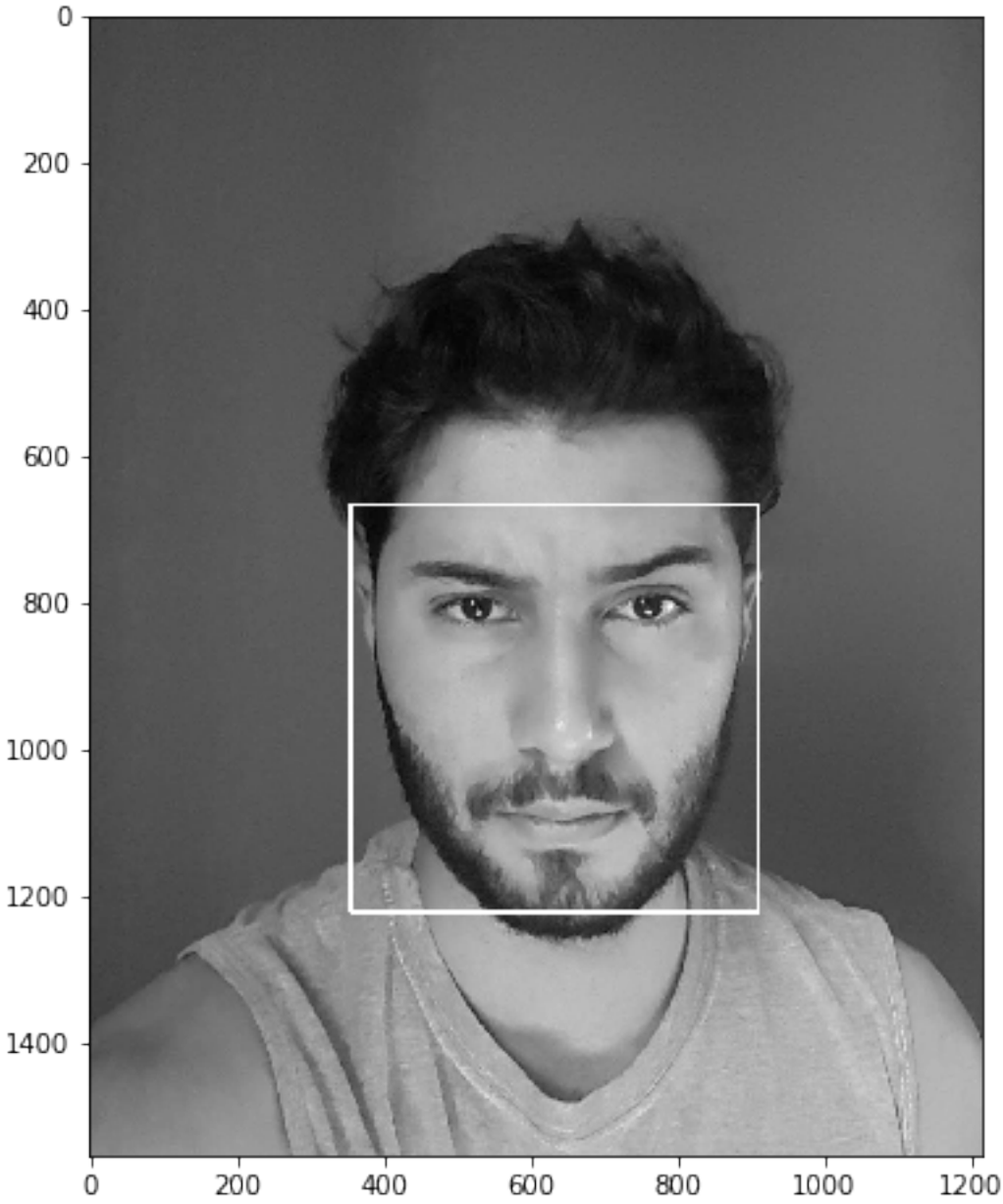
[25]:
```
plt.figure(figsize=(12,8))
plt.imshow(mag, cmap="gray")
plt.show()
```



Detectando o rosto na imagem

[26]:
```
face_detect = dlib.get_frontal_face_detector()

rects = face_detect(gray, 1)

for (i, rect) in enumerate(rects):
    (x, y, w, h) = face_utils.rect_to_bb(rect)
    cv2.rectangle(gray, (x, y), (x + w, y + h), (255, 255, 255), 3)
```

```
plt.figure(figsize=(12,8))
plt.imshow(gray, cmap='gray')
plt.show()
```



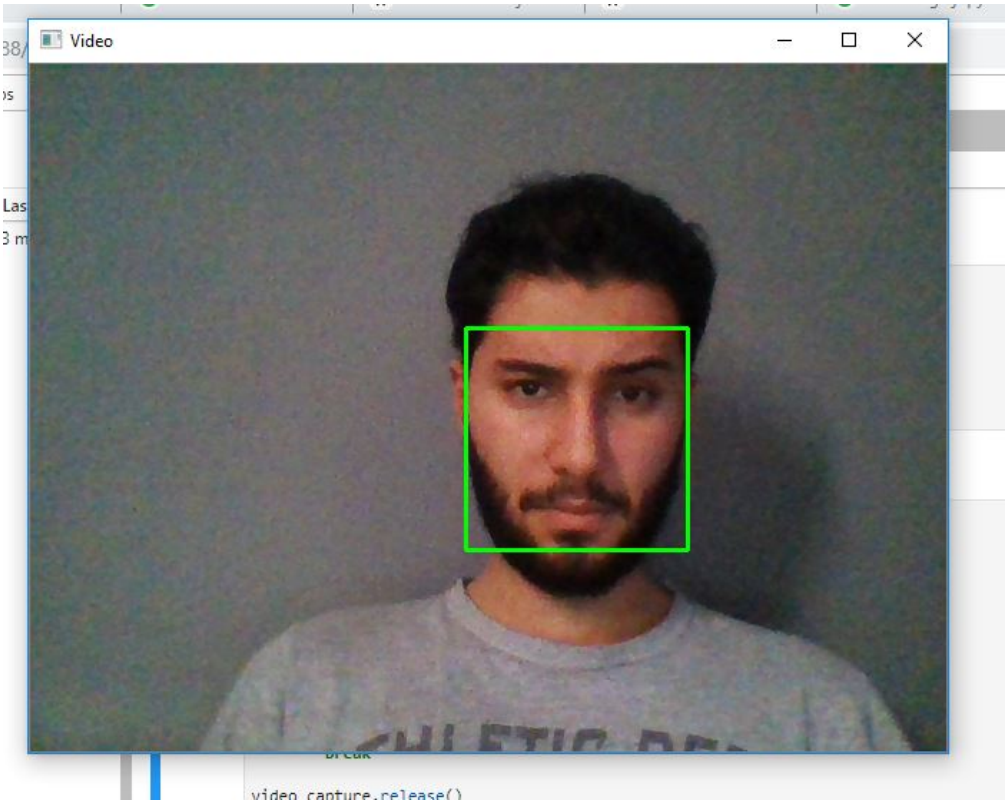
Função de detecção de faces com o HOG

[27]:
```
def detect_faces_hog(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    rects = face_detect(gray, 1)

    for (i, rect) in enumerate(rects):
        (x, y, w, h) = face_utils.rect_to_bb(rect)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Utilizando a web-cam como input

[28]:
```
video_capture = cv2.VideoCapture(0)

while True:

    ret, frame = video_capture.read()

    detect_faces_hog(frame)

    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```



title

3 Convolutional Neural Network

É um método pré-treinado de Redes Neurais Convolucionais que para detecção de faces.

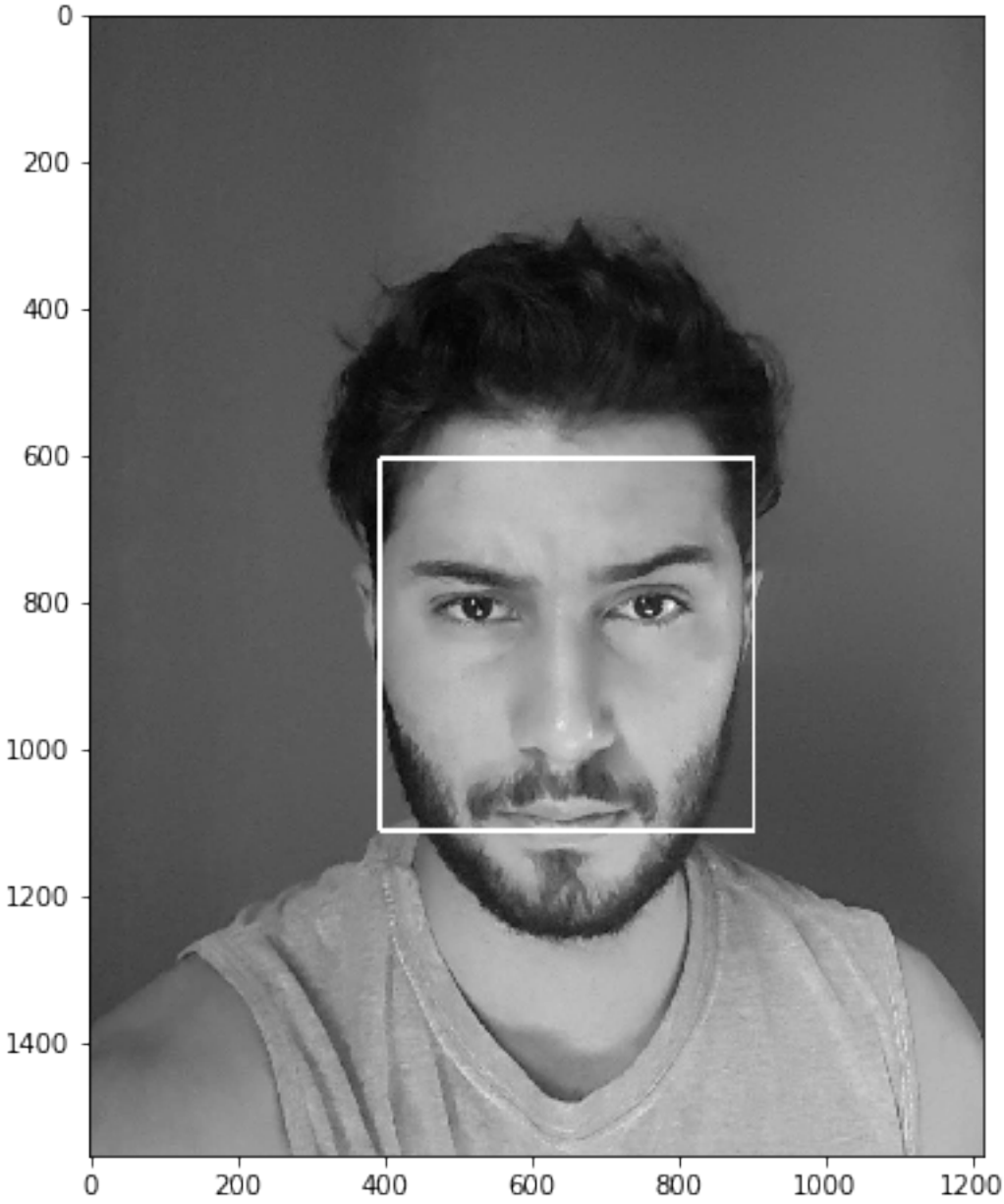
Carregando os pesos

[30]:
```
dnnFaceDetector = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.
↪dat")
```

Detectando o rosto

[31]:
```
gray = cv2.imread('selfie2.jpg', 0)

rects = dnnFaceDetector(gray, 1)

for (i, rect) in enumerate(rects):

    x1 = rect.rect.left()
    y1 = rect.rect.top()
    x2 = rect.rect.right()
    y2 = rect.rect.bottom()

    # Rectangle around the face
```

```
    cv2.rectangle(gray, (x1, y1), (x2, y2), (255, 255, 255), 3)

plt.figure(figsize=(12,8))
plt.imshow(gray, cmap='gray')
plt.show()
```

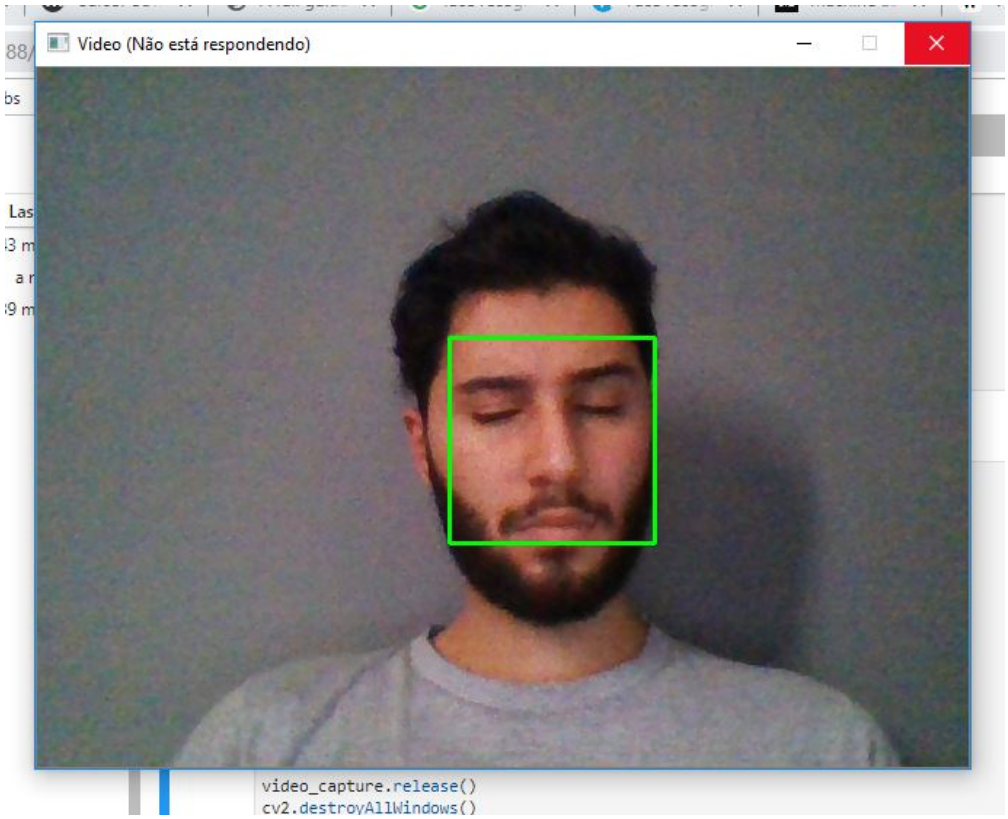


Função para detecção de daces CNN

[32]:
```
def deteccao_faces_cnn(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = dnnFaceDetector(gray, 1)

    for (i, rect) in enumerate(rects):

        x1 = rect.rect.left()
        y1 = rect.rect.top()
        x2 = rect.rect.right()
        y2 = rect.rect.bottom()

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

Usando a web-cam como input

[ ]:
```
video_capture = cv2.VideoCapture(0)
flag = 0

while True:

    ret, frame = video_capture.read()

    deteccao_faces_cnn(frame)

    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```



title

Todos os classificadores reconhecem muito bem rostos, praticamente sem erros em ambiente controlado. A principal diferença é o tempo de processamento dos algoritmos que poderá ser observado melhor com as métricas do vídeo e da web-cam.

3.1 Funções para todos os Detectores

[2]:
```
def detect_faces_cascade(frame, detector):

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    rostos = detector.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

    faces = []

    for (x, y, w, h) in rostos:
        faces.append([x, y, w, h])
```

```
    return faces


def detect_faces_hog(frame, detector):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 1)

    faces = []

    for (i, rect) in enumerate(rects):
        faces.append(list(face_utils.rect_to_bb(rect)))
    return faces

def deteccao_faces_cnn(frame, detector):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 1)

    faces = []

    for (i, rect) in enumerate(rects):

        x = rect.rect.left()
        y = rect.rect.top()
        w = rect.rect.right()
        h = rect.rect.bottom()

        faces.append([x, y, w, h])

    return faces
```

3.2 Vídeo e web-cam

Para a utilização da web cam não foi alterado a resolução padrão do OPENCV, 640x480.

[5]:
```
from datetime import datetime
font = cv2.FONT_HERSHEY_SIMPLEX

def deteccao_wc_video(input_style = 1,  detector_cod = 1, arquivo_vid = "-",␣
 ↪salvar_rosto = False, nome_rosto = ""):

    # carregando o detector de acordo com o código passado
    if detector_cod == 1:
        cascPath = "C:/Users/Henrique/Anaconda3/Lib/site-packages/cv2/data/
 ↪haarcascade_frontalface_default.xml"
        detector = cv2.CascadeClassifier(cascPath)
    elif detector_cod == 2:
```

```
        detector = dlib.get_frontal_face_detector()
    elif detector_cod == 3:
        detector = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.
↪dat")

    # habilitando o input escolhido
    if input_style == 1:
        video_capture = cv2.VideoCapture(0)
    elif input_style == 2:
        video_capture = cv2.VideoCapture(arquivo_vid)
        if (video_capture.isOpened()== False):
            return 0

    # variáveis para a contagem de frames
    seg_anterior = datetime.now().second
    qtd_frames = 0
    cont_frames = 0

    imagem = 1

    # loop de execucao
    while True:

        # lendo o próximo frame  e a flag de validação de leitura
        ret, frame = video_capture.read()

        if ret == True:

            # utiliza o detector
            if detector_cod == 1:
                faces = detect_faces_cascade(frame, detector)
            elif detector_cod == 2:
                faces = detect_faces_hog(frame, detector)
            elif detector_cod == 3:
                faces = deteccao_faces_cnn(frame, detector)

            # desenhar o retangulo do rosto
            for x, y, w, h in faces:

                # salvar imagem para o treinamento
                if salvar_rosto:
                    cv2.imwrite("datasets/" + nome_rosto + "/" + nome_rosto +␣
↪str(imagem) + ".jpg", frame[y:y+h, x:x+w])
                    if imagem == 35:
                        video_capture.release()
                        cv2.destroyAllWindows()
                        return 1
```

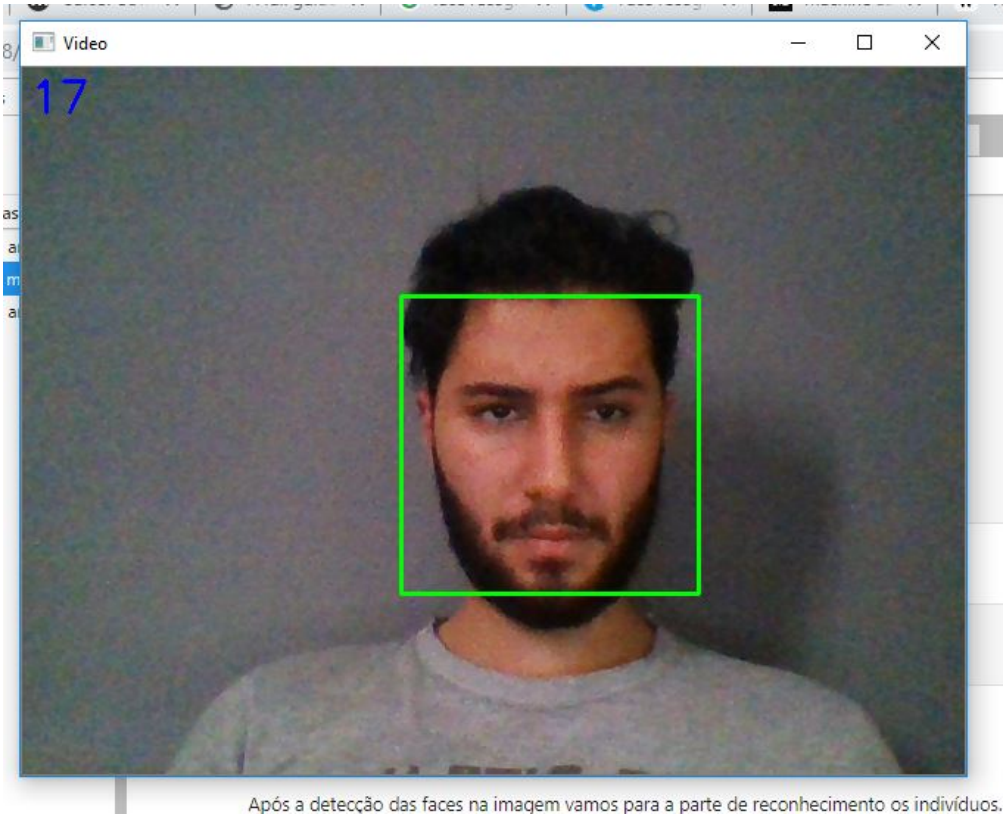
```
                    imagem += 1

                if detector_cod == 3:
                    cv2.rectangle(frame, (x, y), (w, h), (0, 255, 0), 2)
                else:
                    cv2.rectangle(frame, (x, y), (x + w,  y + h), (0, 255, 0),␣
↪2)

            #contador de frames
            cont_frames += 1

            # segundo atual
            seg =  datetime.now().second

            # atualizar contagem de frames
            if seg_anterior != seg:
                seg_anterior = seg
                qtd_frames = cont_frames
                cont_frames = 0

            # adiciona  o contador de frames
            cv2.putText(frame, str(qtd_frames),(5,30), font, 1,(255,0,0),2)
            cv2.imshow('Video', frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break

    video_capture.release()
    cv2.destroyAllWindows()
    return 1
```

3.2.1 Executando o Detector

Com o Cascade Classifier

[10]:
```
# input_style: 1 = web-cam, 2 = arquivo de video
# detector_cod: 1 = Cascade, 2 = HOG, 3 = CNN
deteccao_wc_video(input_style = 1, detector_cod = 2,  salvar_rosto = True,␣
↪nome_rosto = "Henrique")
```

[10]: 1

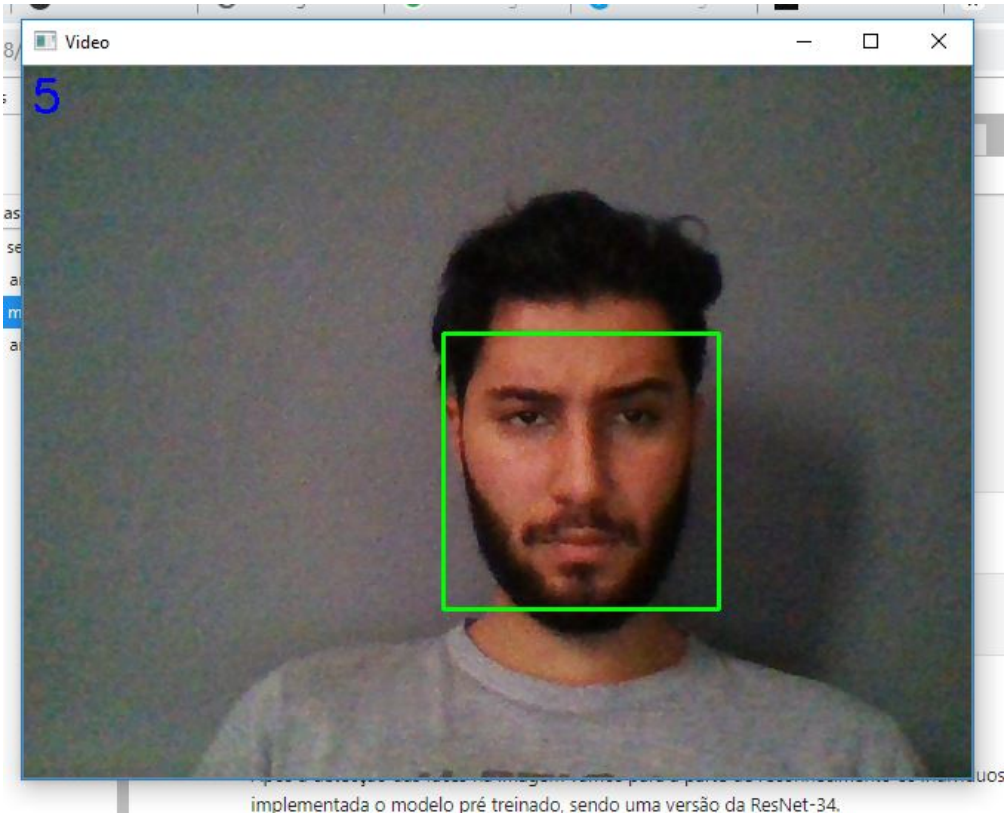


title

Dentre os algoritmos foi o Cascade Classifier foi o mais eficiente em ambientes controlados e quase não apresentou erros. Teve uma taxa de 17 frames por segundo identificando com somente 1 pessoa no vídeo.

HOG

[ ]:
```
# input_style: 1 = web-cam, 2 = arquivo de video
# detector_cod: 1 = Cascade, 2 = HOG, 3 = CNN
deteccao_wc_video(input_style = 1, detector_cod = 2)
```

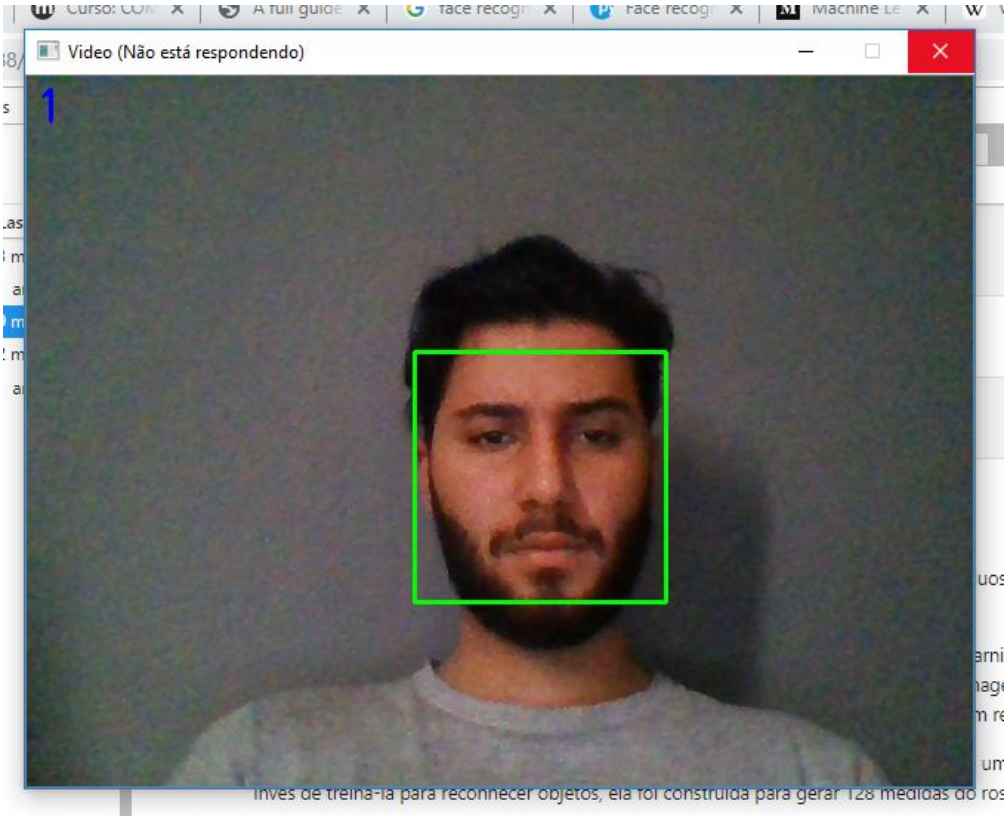


title

O algoritmo HOG teve uma excelente performance de detecção, com quase nenhum erro. Mas diferente do Cascade Classifier a média de frames foi de 5 FPS.

CNN

[ ]:
```
# input_style: 1 = web-cam, 2 = arquivo de video
# detector_cod: 1 = Cascade, 2 = HOG, 3 = CNN
deteccao_wc_video(input_style = 1, detector_cod = 3)
```



title

O CNN foi o detector que menos apresentou falsos positivos porêm com a menor eficiência dentre todos os algoritmos. Levou cerca de 30 segundos para processar cada frame, então é necessário uma máquina com grande poder de processamento.

4 Reconhecimento de Faces

Após a detecção das faces na imagem vamos para a parte de reconhecimento os indivíduos. Para isso utilizaremos a biblioteca face_recognition que já traz implementada o modelo pré treinado de reconhecimento, sendo uma versão da ResNet-34.

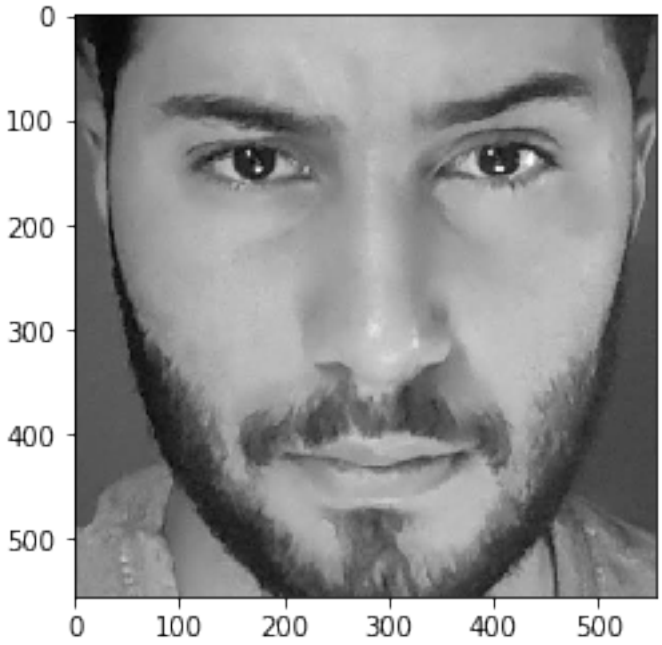
O ResNet-34 é uma Rede Neural Convolucional apresentada no paper Deep Residual Learning for Image Recognition pelo Kaiming He, onde ele apresenta uma maneira de treinamento residual para facilitar o treinamento para reconhecimento de imagens. Nesse método ele reformula as camadas para receber referências de outras camadas de input, ao contrário de aprender através de funções sem referência.

O modelo implementado na biblioteca 'face_recognition' foi treinado por Davis King em uma base de dados de aproximadamente 3 milhões de imagens e, ao invés de treiná-la para reconhecer objetos, ela foi construída para gerar 128 medidas do rosto humano.

Para a imagem abaixo foram identificado essas 128 medidas.

[413]:
```
plt.figure(figsize=(4,4))
plt.imshow(cv2.cvtColor(face, cv2.COLOR_BGR2GRAY), cmap="gray")
plt.show()
```

```
print(face_recognition.face_encodings(face))
```




```
[array([-0.1679188 ,  0.17989676,  0.04687099, -0.06977943, -0.09041418,
        0.07881704, -0.01151275, -0.08020727,  0.21354845, -0.15110849,
        0.25033399,  0.05917898, -0.14450884, -0.05953503, -0.01793103,
        0.11419031, -0.16621715, -0.14848892, -0.08891002, -0.06253053,
        0.03707078,  0.06890924,  0.002906  ,  0.02287783, -0.11091387,
       -0.37076396, -0.07505058, -0.09265973,  0.10694541, -0.14356299,
       -0.04581314, -0.06061215, -0.16111469, -0.02189315, -0.01230724,
        0.05811262, -0.00099564, -0.11495323,  0.14597414, -0.02609702,
       -0.11759105,  0.01820884,  0.06558762,  0.31950337,  0.17907068,
       -0.02690672,  0.01392409, -0.08795253,  0.126509  , -0.2067585 ,
        0.13725516,  0.10575123,  0.0626056 ,  0.0646778 ,  0.13840225,
       -0.09124059, -0.02407597,  0.12519203, -0.16766149,  0.09907741,
        0.07220753,  0.06612866, -0.05233006, -0.0825004 ,  0.21337169,
        0.15196581, -0.13441496, -0.14938441,  0.10926422, -0.11334927,
       -0.04793226, -0.02010375, -0.11431266, -0.18273234, -0.16158433,
        0.09345284,  0.4247095 ,  0.21947613, -0.18444464,  0.04794114,
       -0.10016592, -0.10025014,  0.06337804,  0.07571232, -0.07193972,
       -0.01206573, -0.07980608,  0.1298252 ,  0.20592959,  0.07779517,
       -0.03932106,  0.25815809, -0.02104623, -0.07603838,  0.01390132,
        0.08321413, -0.16099648,  0.02500054, -0.13304207, -0.07323226,
        0.07870162, -0.07503   , -0.02569129,  0.13824335, -0.20088905,
        0.2055788 ,  0.0072318 ,  0.04210716,  0.06219536,  0.00525997,
```

```
-0.10128927,  0.01532075,  0.1285013 , -0.20607835,  0.20134677,
 0.09571505,  0.07024553,  0.1769221 ,  0.08189276,  0.05289037,
 0.08604506, -0.02774211, -0.15537158, -0.07812595,  0.01027466,
-0.03992386,  0.08492982,  0.06583021])]
```

Não sabemos o que essas medidas represantam, mas como são utilizados as mesmas métricas para todas as imagens conseguimos utilizá-las para a classificação.

Os passos que utilizaremos para o reconhecimento de pessoas:

Através dos métodos Cascade Classifier, HOG ou CNN iremos identificar e separar os rostos nas imagens (método já realizado anteriormente nesse documento).

Utilizaremos o 'face_recognition' para mensurar as 128 medidas do rosto.

Vamos salvar uma base de treinamento com as medidas já mensuradas.

A partir da base de treinamento vamos verificar com quais medidas a nova pessoa mais se assemelha para a identificação.

4.0.1 Gerando a base de treinamento

Para fazer o cálculo das medidas precisamos criar uma base de dados das pessoas que queremos identificar. Utilizaremos a função criada 'deteccao_wc_video' que já faz a detecção de rostos e salvaremos os 30 frames que possuem rostos. Os parâmetros que serão utilizados para salvar as imagens: salvar_rosto = True, nome_rosto = "nome da pessoa".

Para que os dados sejam consistentes é necessário que tenha somente 1 pessoa nos frames, já que não foi feito nenhuma tratativa para casos com mais de um indivíduo, assim deixando os dados inconsistentes.

[ ]:
```
deteccao_wc_video(input_style = 1, detector_cod = 2, arquivo_vid = "video1.
↪mp4", salvar_rosto=True, nome_rosto="Henrique")
```

Para organizar as bases de imagens foi definida a seguinte estrutura:

- datasets
  - pessoa1
    - foto1
    - foto2
    - ...
    - foto 30
  - pessoa2
    - foto1
    - foto2
    - ...
    - foto 30
  - ...
  - pessoaN
    - foto1
    - foto2
    - ...
    - foto 30

Após a preparação das bases vamos calcular suas medidas e salvar em um arquivo.

[20]:
```
def medidas_imagens_base():
    # pasta que está localizado a imagem das pessoas
    pasta = "datasets"
    jpgs = []
    nomes = []

    # pega todos os caminhos da pasta datasets, que representam as pessoas
    caminhos = [nome for nome in os.listdir(pasta)]

    # para cada caminho encontrado
    for caminho in caminhos:

        # pego todos os arquivos
        arquivos = [pasta +"/"+ caminho +"/"+ nome for nome in os.listdir(pasta␣
↪+ "/" + caminho)]

        # adiciono na lista de todos as imagens
        aux_jpgs = [arq for arq in arquivos if arq.lower().endswith(".jpg")]
        jpgs = jpgs + aux_jpgs

        # adiciono o nome  da  pessoa de cada imagem
        aux_nomes = [caminho] * len(aux_jpgs)
        nomes = nomes + aux_nomes

    # criar os encodings de cada rosto para treinamento
    encodings = []
    for i in range(len(jpgs)):

        # carrega a imagem e transforma para rgb
        img = cv2.imread(jpgs[i])
        rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # encontra os as 128 medidas de cada imagem
        encoding = face_recognition.face_encodings(rgb)

        if (len(encoding) > 0):
            encodings.append(encoding[0])

    # cria um dicionário com as medidas e sua classificação
    data = {"encodings": encodings, "names": nomes}

    # salva o dicionário em um arquivo
    f = open("encodings/encoding", "wb")
    f.write(pickle.dumps(data))
    f.close()
```

[24]:
```
medidas_imagens_base()
```

4.0.2 Reconhecer os Rostos

Para reconhecer os rostos vamos pegar as medidas já salvas no arquivo 'encoding' e iremos comparar com as novas medidas. A identificação implementada pela biblioteca 'face_recognition' é o princípio do KNN, onde é medido a distância euclidiana entre a nova imagem e a base de treinamento e é passado um threshold(raio) de aceitação para definir os vizinhos. Após a detecção dos vizinhos, por meio do voto majoritário, classificaremos a nova imagem.

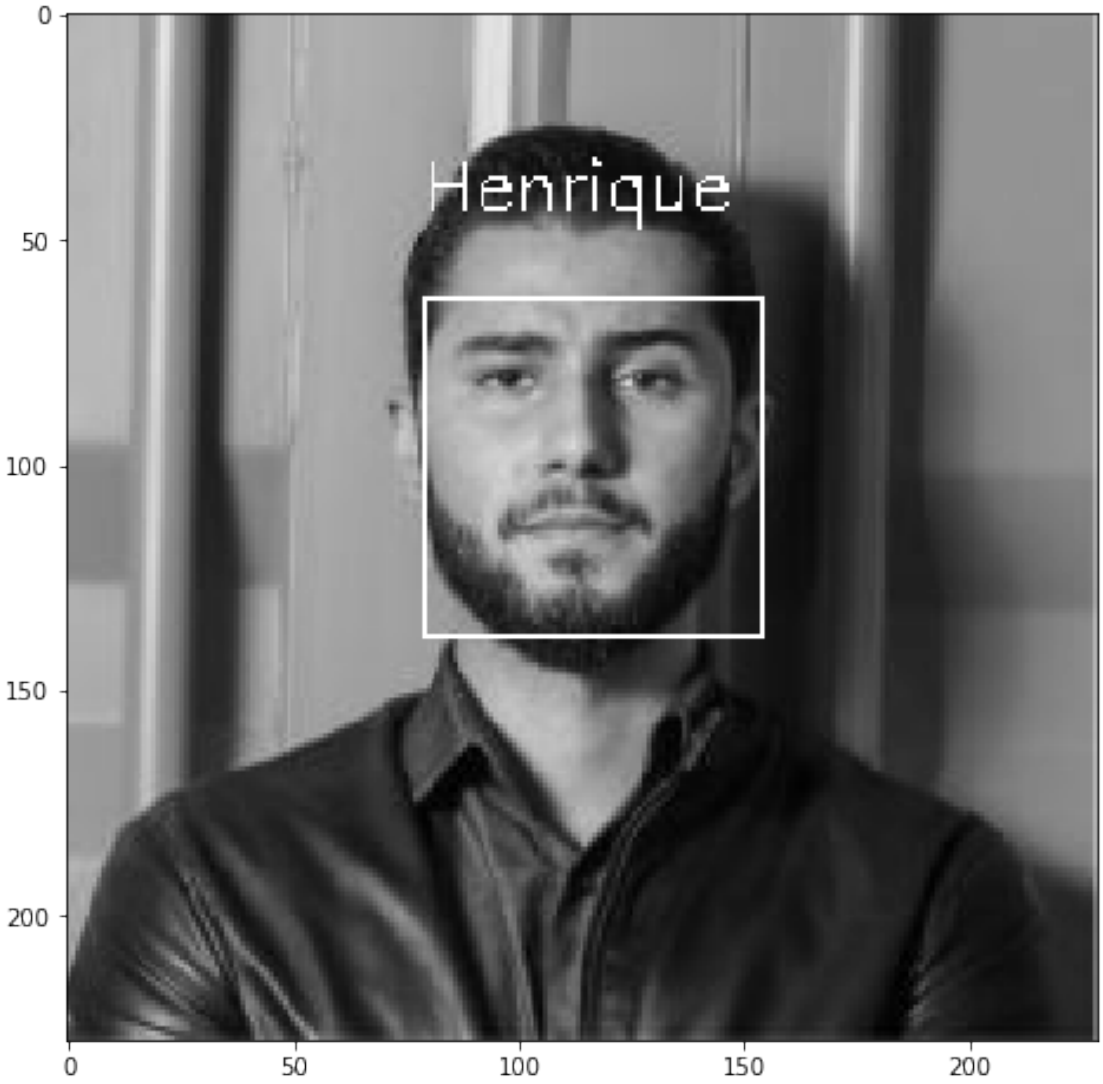
[13]:
```
def reconhecer_rosto(face, data):
    # definir as medidas do novo rosto
    encoding = face_recognition.face_encodings(face)

    nomes = []

    if len(encoding) > 0:
        # define quais os rostos estão na distância de tolerância definida
        matches = face_recognition.compare_faces(np.array(data["encodings"]),␣
↪np.array(encoding[0]), tolerance = 0.5)

        # adiciona todas as pessoas próximas da imagem
        for i in range (len(matches)):
            if matches[i] == True:
                nomes.append(data["names"][i])

    if len(nomes) == 0: nomes = ["desconhecido"]


    # cria um histograma das pessoas reconhecidas
    nomes_unique = set(nomes)
    qtd_matches = []
    for i in nomes_unique:
        qtd_matches.append([i, nomes.count(i)])

    # seleciona a pessoa de maior ocorrÊncia
    maior = qtd_matches[0]
    for i in qtd_matches:
        if maior[1] < i[1]:
            maior = i

    # retorna o nome da pessoa se a pessoa for reconhecida
    return maior[0]
```

4.0.3 Classificando uma pessoa

Utilizaremos o trabalho realizado acima para identificar uma pessoa da base de treinamento

[12]:
```
# carregar as dimensões treinadas
data = pickle.loads(open("encodings/encoding", "rb").read())
```

```
# carregar imagem para classificacao e converter para RGB
image = cv2.imread("capturar.jpg")
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# transformar para tons de cinza para detecção do rosto
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# utilizando HOG para detecção
face_detect = dlib.get_frontal_face_detector()
rects = face_detect(gray, 1)

# separar as medidas do rosto encontrado
for (i, rect) in enumerate(rects):
    recta = list(face_utils.rect_to_bb(rect))

# separar rosto
(x, y, w, h) = recta
face = rgb[y:y+h, x:x+w]

reconhecimento = reconhecer_rosto(face, data)

cv2.rectangle(gray, (x, y), (x + w, y + h), (255, 0, 0), 1)
cv2.putText(gray, reconhecimento, (x, y-20), cv2.FONT_HERSHEY_SIMPLEX,0.5,␣
 ↪(255, 0, 0), 1)

plt.figure(figsize=(12,8))
plt.imshow(gray, cmap="gray" )
plt.show()
```



4.0.4 Atualizando a função 'deteccao_wc_video' para o reconhecimento de faces em tempo real

Vamos adicionar as funções de reconhecimento de face para o vídeo e a web cam.

[4]:
```
from datetime import datetime
font = cv2.FONT_HERSHEY_SIMPLEX

def deteccao_wc_video_reconhecimento(input_style = 1,  detector_cod = 1,␣
 ↪arquivo_vid = "-", salvar_rosto = False, nome_rosto = "",␣
 ↪salvar_video=False, nome_saida=""):

    # carregar as dimensões treinadas
    data = pickle.loads(open("encodings/encoding", "rb").read())
```

```
    # carregando o detector de acordo com o código passado
    if detector_cod == 1:
        cascPath = "C:/Users/Henrique/Anaconda3/Lib/site-packages/cv2/data/
↪haarcascade_frontalface_default.xml"
        detector = cv2.CascadeClassifier(cascPath)
    elif detector_cod == 2:
        detector = dlib.get_frontal_face_detector()
    elif detector_cod == 3:
        detector = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.
↪dat")

    # habilitando o input escolhido
    if input_style == 1:
        video_capture = cv2.VideoCapture(0)
    elif input_style == 2:
        video_capture = cv2.VideoCapture(arquivo_vid)
        if (video_capture.isOpened()== False):
            return 0

    # variáveis para a contagem de frames
    seg_anterior = datetime.now().second
    qtd_frames = 0
    cont_frames = 0

    imagem = 1

    if salvar_video: out = cv2.VideoWriter(nome_saida,cv2.
↪VideoWriter_fourcc('M','J','P','G'), 30, (640,480))

    # loop de execucao
    while True:

        # lendo o próximo frame  e a flag de validação de leitura
        ret, frame = video_capture.read()

        if ret == True:

            # utiliza o detector
            if detector_cod == 1:
                faces = detect_faces_cascade(frame, detector)
            elif detector_cod == 2:
                faces = detect_faces_hog(frame, detector)
            elif detector_cod == 3:
                faces = deteccao_faces_cnn(frame, detector)

            # desenhar o retangulo do rosto
```

```
            for x, y, w, h in faces:

                # salvar imagem para o treinamento
                if salvar_rosto:
                    cv2.imwrite("datasets/" + nome_rosto + "/" + nome_rosto +␣
↪str(imagem) + ".jpg", frame[y:y+h, x:x+w])
                    if imagem == 35:
                        video_capture.release()
                        cv2.destroyAllWindows()
                        return 1
                    imagem += 1



                face = frame[y:y+h, x:x+w]
                reconhecimento = reconhecer_rosto(face, data)

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, reconhecimento, (x, y-20), cv2.
↪FONT_HERSHEY_SIMPLEX,1, (255, 0, 0), 2)

            #contador de frames
            cont_frames += 1

            # segundo atual
            seg =  datetime.now().second

            # atualizar contagem de frames
            if seg_anterior != seg:
                seg_anterior = seg
                qtd_frames = cont_frames
                cont_frames = 0

            # adiciona  o contador de frames
            cv2.putText(frame, str(qtd_frames),(5,30), font, 1,(255,0,0),2)
            cv2.imshow('Video', frame)

            if salvar_video: out.write(frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break

    if salvar_video: out.release()

    video_capture.release()
```

```
    cv2.destroyAllWindows()
    return 1
```

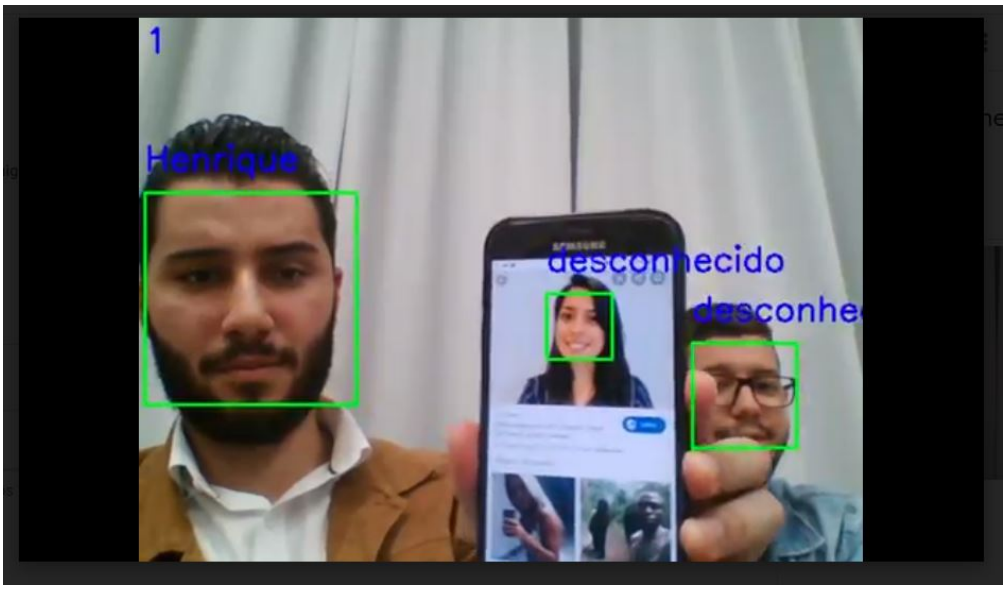
Por conta de poder computacional foi testado o método de classificação somente com os detectores Cascade Classifier e HOG. O reconhecimento ocorreu muito bem mesmo com algumas obstruções mínimas no rosto e como o método de reconhecimento é o KNN, reostos não treinados aparecem como desconhecido.

4.0.5 Teste com o Cascade Classifier

[26]:
```
# input_style: 1 = web-cam, 2 = arquivo de video
# detector_cod: 1 = Cascade, 2 = HOG, 3 = CNN
deteccao_wc_video_reconhecimento(input_style = 2, detector_cod = 1,␣
 ↪arquivo_vid="video.avi" , salvar_video=True, nome_saida="reconhecimento_cc.
 ↪avi")
```

[26]:
```
1
```

vídeo completo: https://drive.google.com/open?id=1gVohwMYdBCgQR_W2TZ4LatwPP1cJDFF7



title

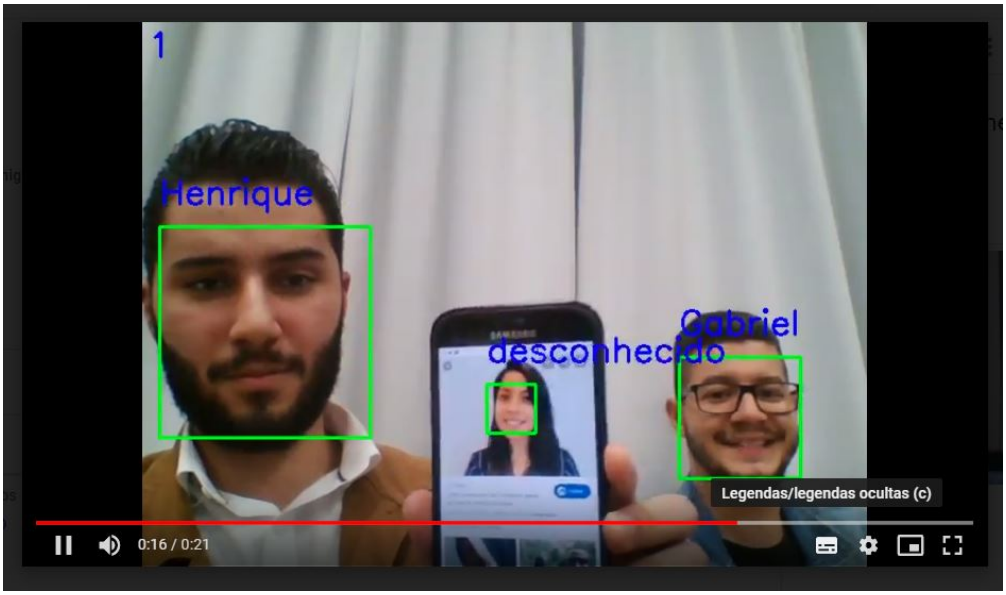
Com obstruções maior o reconhecimento se mostrou falho. Todos as pessoas foram treinadas sem óculos, as pessoas que utilizaram óculos nos testes demonstraram mais inconsistências nas classificações.

4.0.6 Teste com o HOG

[ ]:
```
# input_style: 1 = web-cam, 2 = arquivo de video
# detector_cod: 1 = Cascade, 2 = HOG, 3 = CNN
```

```
deteccao_wc_video_reconhecimento(input_style = 2, detector_cod = 2,␣
↪arquivo_vid="video.avi" , salvar_video=True, nome_saida="reconhecimento_hog.
↪avi")
```

vídeo completo: https://drive.google.com/open?id=122VfcvjBsbeJzU0zsvIaENX7Jb9xZzqr



title

5 Melhorias futuras

- Melhorar a base de treinamento e fazer um data augmentation para extrair features que representam melhor cada indivíduo.
- Testar outros algoritmos de classificação(Regressão Logística, SVM...) e determinar qual é o mais eficiênte para trabalhar com esses dados.
- Utilizar GPU.