Michel André Lima Vinagreiro

Explicação Detalhada dos Programas e Processos Realizados

Fatec Santo André

Dezembro de 2019

O objetivo deste tutorial é explicar, para todos os envolvidos no projeto de reconhecimento facial, como se deram todos os processos de tratamento inicial das imagens, detecção e recorte das faces nas imagens, montagem do banco de vetores que representam as imagens, separação dos conjuntos de treino e teste da rede CNN que classifica a imagem recortada, treinamento da rede CNN, montagem dos programas de treinamento da rede, montagem do banco de vetores, extração de *frames* dos vídeos dos usuários e programa de identificação de face de usuário embarcado.

A seguir, será explicado de forma cronológica e detalhada, todos os processos realizados.

1. Extração dos *frames* a partir dos arquivos de vídeo.

O passo inicial se dá pela extração dos *frames* a partir dos arquivos de vídeo disponibilizado pelos usuários. Os *frames* são armazenados em diretórios para posterior utilização em outros processos.

O programa *lervideo.py* captura os *frames* e armazena em um diretório:

```
@author: michel
'''
import numpy as np
import cv2

cap = cv2.VideoCapture('henrique_novo.mp4')
imcout=0
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        frame=cv2.flip(frame,-1)
        frame=cv2.flip(frame,0)
        frameret=cv2.resize(frame,
(frame.shape[0],frame.shape[1]),interpolation=cv2.INTER_LANCZOS4)
        for i in range(frame.shape[0]):
            for j in range(frame.shape[1]):
                frameret[j][i]=frame[i][j]
        frameret=cv2.flip(frameret,1)
        cv2.imwrite("henrique/img"+str(imcout)+".jpg",frameret)
        imcout+=1
    else:
        break

cap.release()
print("terminou!!!!, imagens salvas!")
```

O exemplo de programa acima lê o arquivo *frame à frame* e salva no diretório *“henrique”* , as imagens são nomeadas de *img0.jpg* à *imgn-1.jpg*, onde *n* é o número total de *frames* que o arquivo de vídeo contém.

1.1. Detecção de faces, recorte e criação do banco de vetores para comparação.

O API *face_recognition* detecta a face em uma imagem, isola a região onde foi detectada a face e aplica à uma rede CNN pré-treinada que produz como saída um vetor utilizado para comparações posteriores.

O programa *treina_face.py* lê as imagens dos diretórios dos três usuários e de outro denominado *intruso*, que não possuirá acesso concedido. As imagens após serem lidas são submetidas a detecção de face, onde, a região é recortada, aplica a rede pré-treinada e o vetor resultante, acompanhado do rótulo (usuário ou intruso) é adicionado ao banco de vetores. Ao final de todo o processo, o banco de vetores é salvo em formato de arquivo de extensão “.pickle” para posterior utilização.

```python
import numpy as np
import face_recognition
import cv2
import pickle
kencoding=[]
knames=[]


for iran in range(3):
    a=cv2.imread("iranilson/img"+str(iran+30)+".jpg",3)

    boxes=face_recognition.face_locations(a,model="cnn")
    encodings=face_recognition.face_encodings(a,boxes)
    for encoding in encodings:
        kencoding.append(encoding)
        knames.append("iranilson")
        print("processando iranilson img"+str(iran))

for hen in range(3):
    a=cv2.imread("henrique/img"+str(hen+155)+".jpg",3)
    boxes=face_recognition.face_locations(a,model="cnn")
    encodings=face_recognition.face_encodings(a,boxes)
    for encoding in encodings:
        print(boxes)
        kencoding.append(encoding)
        knames.append("henrique")
    print("processando henrique img"+str(hen))

for wand in range(3):
    a=cv2.imread("wands/img"+str(wand+670)+".jpg",3)
    boxes=face_recognition.face_locations(a,model="cnn")
    encodings=face_recognition.face_encodings(a,boxes)
    for encoding in encodings:
        kencoding.append(encoding)
        knames.append("wands")
    print("processando wands img"+ str(wand))

for y in range(200):
    a=cv2.imread("FEI_frontal/"+str(y+1)+"a.jpg",3)
    print("imagem"+str(y+1)+"a")
    boxes=face_recognition.face_locations(a,model="cnn")
    encodings=face_recognition.face_encodings(a,boxes)
    for encoding in encodings:
        print(boxes)
        kencoding.append(encoding)
        knames.append("intruso")
        print("processando Intruso_fei")
    a=cv2.imread("FEI_frontal/"+str(y+1)+"b.jpg",3)
    print("imagem"+str(y+1)+"b")
    boxes=face_recognition.face_locations(a,model="cnn")
    encodings=face_recognition.face_encodings(a,boxes)
    for encoding in encodings:
        print(boxes)
        kencoding.append(encoding)
        knames.append("intruso")
        print("processando Intruso_fei")

data={"encodings":kencoding,"names":knames}
f=open("enc_TCC.pickle","wb")
f.write(pickle.dumps(data))
f.close()
```

1.2. Montagem dos conjuntos de dados para treinamento e testes a partir das faces recortadas.

Os APIs *tensorFlow* e *keras* são utilizados para confeccionar e treinar a rede CNN utilizada para classificar as faces recortadas. O programa *keras_dataset_make.py* é utilizado para recortar as faces presentes nas imagens e armazenar em uma estrutura de diretórios utilizada pelos APIs para treinar e testar a rede, a estrutura de diretórios, para a confecção da rede que classifica as faces como *"henrique"* ou *"intruso"* será a seguinte:

dataset_keras
- *train*
  - *herinque*
  - *intruso*
- *test*
  - *henrique*
  - *intruso*

Outras duas redes idênticas são utilizadas para classificar as faces dos outros dois usuários como pertencentes aos usuários ou intrusas.

```
import numpy as np
import face_recognition
import cv2

for i in range(400):
    a=cv2.imread("henrique/img"+str(i)+".jpg",3)
    boxes=face_recognition.face_locations(a,model="cnn")

    y0=int(boxes[0][0])
    y1=int(boxes[0][2])
    x0=int(boxes[0][3])
    x1=int(boxes[0][1])
    b=a[y0:y1,x0:x1]
    b=cv2.resize(b,(224,224),interpolation=cv2.INTER_LANCZOS4)
    cv2.imwrite("dataset_keras/train/henrique/"+str(i+1)+".jpg",b)
    print(str(i+1)+"...")
print("henrique salvo...")
u=1
for y in range(200):
    a=cv2.imread("FEI_frontal/"+str(y+1)+"a.jpg",3)
    boxes=face_recognition.face_locations(a,model="cnn")
    y0=int(boxes[0][0])
    y1=int(boxes[0][2])
    x0=int(boxes[0][3])
    x1=int(boxes[0][1])
    b=a[y0:y1,x0:x1]
    b=cv2.resize(b,(224,224),interpolation=cv2.INTER_LANCZOS4)
    cv2.imwrite("dataset_keras/train/intruso/"+str(u)+".jpg",b)
    u+=1
    a=cv2.imread("FEI_frontal/"+str(y+1)+"b.jpg",3)
    boxes=face_recognition.face_locations(a,model="cnn")
    y0=int(boxes[0][0])
    y1=int(boxes[0][2])
    x0=int(boxes[0][3])
    x1=int(boxes[0][1])
    b=a[y0:y1,x0:x1]
    b=cv2.resize(b,(224,224),interpolation=cv2.INTER_LANCZOS4)
    cv2.imwrite("dataset_keras/train/intruso/"+str(u)+".jpg",b)
    u+=1
    print(str(y+1)+"...")
print("intruso salvo...")
```

Após recortadas, todas as imagens são redimensionadas para a resolução padrão de 224 por 224 *pixels.* As imagens utilizadas para o treinamento da classe *"intruso"* são provenientes do *dataset FEI_database*, disponibilizado publicamente pela instituição de ensino FEI de São Bernardo do Campo. As imagens utilizadas para os testes da mesma classe são provenientes do *dataset labeded faces in the wild* e são selecionadas de forma randômica. Ao todo 400 imagens de treinamento para cada classe são utilizadas e 100 imagens de teste para cada classe.

1.3. Arquitetura da rede.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import tensorflow as tf


img_width, img_height = 224, 224

train_data_dir = 'dataset_keras/train'
validation_data_dir = 'dataset_keras/test'
nb_train_samples = 800
nb_validation_samples = 160
epochs = 25
batch_size = 16

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (2, 2), input_shape = input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Conv2D(32, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Conv2D(64, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss ='binary_crossentropy',
                    optimizer ='RMSprop',
                  metrics =['accuracy'])

train_datagen = ImageDataGenerator(
                rescale = 1. / 255,
                 shear_range = 0.2,
                  zoom_range = 0.2,
            horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1. / 255)

train_generator = train_datagen.flow_from_directory(train_data_dir,
                           target_size =(img_width, img_height),
                      batch_size = batch_size, class_mode ='binary')

validation_generator = test_datagen.flow_from_directory(
                                    validation_data_dir,
                    target_size =(img_width, img_height),
          batch_size = batch_size, class_mode ='binary')

model.fit_generator(train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
```

Como mencionado anteriormente, as imagens de entrada possuem a resolução padrão de 224 por 224 *pixels.* A primeira camada convolucional 32 *kernels* de 2 por 2, em seguida é aplicada a redução de dimensionalidade *maxpooling* de 2 por 2, a função de ativação é do tipo linear retificada (*relu*).

A segunda camada convolucional possui as mesmas características da primeira.

A terceira camada convolucional possui 64 *kernels* de 2 por 2, em seguida *maxpooling* de 2 por 2, a função de ativação é do tipo linear retificada.

A primeira camada totalmente conectada possui 64 neurônios e a função de ativação é do tipo linear retificada, em seguida é aplicado o recurso de *dropout*, que possui a função de evitar a ocorrência de *overfitting*.

A camada de saída é composta por um neurônio e a função de ativação é do tipo *sigmóide.*

Como medida de erro da rede é utilizada a entropia cruzada binária.

Após concluído o treinamento da rede, a arquitetura da rede, juntamente com os pesos treinados, são salvos em formato de arquivo de extensão ".h5", que possibilita que a rede seja utilizada posteriormente por outra aplicação.

Para treinar a rede foi utilizado o recurso de computação em nuvem disponibilizado gratuitamente pela empresa *google* de nominado *colab notebook*, que dispõe de uma interface em linguagem *python 3.6* e uma *GPU Nvidia tesla K80*, que acelera enormemente o tempo de treinamento.

1.3. Faces de exemplo para cálculo de distância.

Três exemplos de faces neutras, um para cada usuário, é utilizada para computar, utilizando a rede pré-treinada disponível na biblioteca *face_recognition*, um vetor que será utilizado no calculo de distância adicional realizado durante a etapa de predição, a função deste cálculo de distância é diminuir a taxa de falsos positivos.

2. Etapa de predição.

O programa de predição utiliza os arquivos gerados para localizar e identificar faces em *frames* capturados por uma câmera que filma o ambiente do interior do automóvel.

A primeira etapa do programa consiste em carregar todos os arquivos gerados durante a etapa de treinamento. A segunda etapa consiste em capturar constantemente *frames* e submetendo ao processo de detecção de faces utilizando a técnica de *HOG.* Após localizada, a região da face é isolada e aplicada a rede pré-treinada que gera um vetor que será comparado com os vetores presentes no banco de vetores. Se o vetor for classificado como pertencente a face de algum dos três usuários do sistema, essa região será recortada e redimensionado para a resolução 224 por 224 *pixels,* que será utilizada como imagem de entrada da rede CNN. Uma vez que a rede CNN classificou a imagem como pertencente a ao mesmo usuário, a distancia euclidiana entre a imagem de referência do usuário e a face é mensurada, e se estiver abaixo de um limiar determinado o acesso é considerado concedido e a saída GPIO a qual a interface relé está conectada é acionada por um tempo de 5 segundos.