Laboratorium 12

Rozpoznawanie osób na zdjęciu za pomocą biblioteki Dlib

Zadanie

1. zadanie weryfikacji tożsamości

Proszę sprawdzić tożsamość swojego zdjęcia paszportowego albo z dowolnego dowodu tożsamości i swojego zdjęcia wykonanego za pomocą kamery internetowej.

Użyjemy wstępnie wytrenowanego modelu ResNet34

Ostatnie warstwy odpowiedzialne za klasyfikację są odcięte od sieci, a pozostają tylko warstwy splotowe, które wyodrębniają kluczowe cechy z obrazu. Wynikiem działania sieci jest zbiór 128 liczb, zwanych deskryptorami. Zatem twarz jest punktem w przestrzeni 128-wymiarowej. Geometrycznie, deskryptory różnych zdjęć jednej osoby są zgrupowane w jednym miejscu w tej przestrzeni, a deskryptory zdjęcia innej osoby są zgrupowane w innym miejscu w przestrzeni. Aby zweryfikować dwie twarze na różnych zdjęciach, musimy znaleźć odległość euklidesową między ich deskryptorami i oszacować bliskość między nimi. Jeśli odległość jest mniejsza niż 0,6, przyjmuje się, że te zdjęcia przedstawiają te same osoby.

Należy wyciągnąć takie deskryptory ze swego zdjęcia w legitymacji (dowodzie tożsamości) oraz z innego dowolnego zdjęcia.

Do pracy potrzebujemy wcześniej wytrenowanych modeli, aby wyróżnić osobę na zdjęciu i usunąć deskryptory ze zdjęcia. Modele te można pobrać ze strony

http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

http://dlib.net/files/dlib_face_recognition_resnet_model_v1.dat.bz2

Pobrane pliki to archiwa, które należy rozpakować i skopiować, na przykład do katalogu ze zdjęciami do rozpoznawania.

Rozważmy przykładową strukturę kodu programu.

```
import dlib
from skimage import io
from scipy.spatial import distance
```

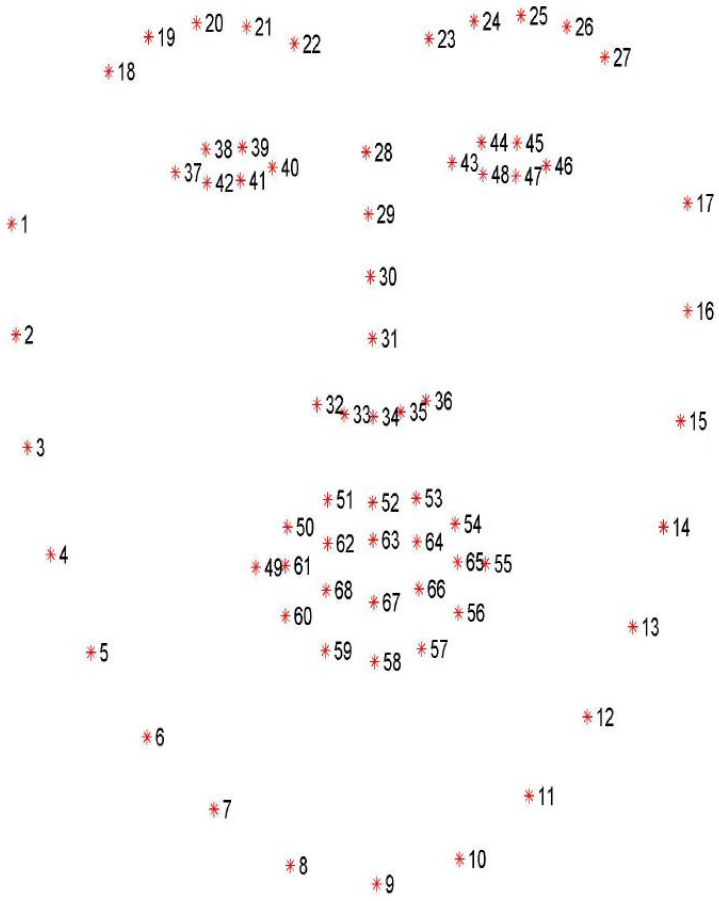
```
sp = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
facerec = dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat
                                         ')
detector = dlib.get_frontal_face_detector()
```

```
img = io.imread('foto.jpg')
win1 = dlib.image_window()
win1.clear_overlay()
win1.set_image(img)
```

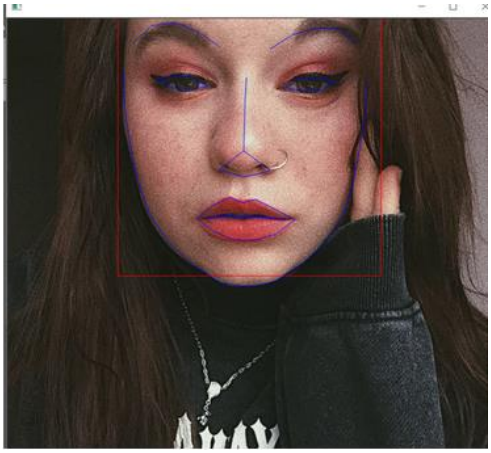
Funkcja dlib.get_frontal_face_detector znajduje na zdjęciu współrzędne wierzchołków prostokąta, w którym znajduje się twarz, za pomocą algorytmów Kazemi'ego i Sullivan'a.

```
dets = detector (img , 1)
for k, d in enumerate ( dets ):
print (" Detection {}: Left : {} Top : {} Right : {} Bottom : {}". format (
k, d. left () , d. top () , d. right () , d. bottom ())) # ramka(prostokont)
shape = sp(img , d)
win1 . clear_overlay ()
win1 . add_overlay (d)
win1 . add_overlay ( shape )
```

Lokalizacja kluczowych punktów



Znalezioną twarz i kluczowe punkty na niej pokazano na zdjęciu



```
Detection 0: Left: 277 Top: 634 Right: 598 Bottom: 955
Detection 0: Left: 161 Top: -52 Right: 546 Bottom: 376
```

```
a =  0.8611627776023943
```

```
import dlib
from skimage import io
from scipy.spatial import distance

sp = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
facerec =
dlib.face_recognition_model_v1('dlib_face_recognition_resnet_model_v1.dat')
detector = dlib.get_frontal_face_detector()

img = io.imread('pic.jpg')
win1 = dlib.image_window()
win1.clear_overlay()
win1.set_image(img)

dets = detector(img, 1)
for k, d in enumerate(dets):
    print("Detection {}: Left: {} Top: {} Right: {} Bottom: {}".format(
        k, d.left(), d.top(), d.right(), d.bottom()))

    shape = sp(img, d)
    win1.clear_overlay()
    win1.add_overlay(d)
    win1.add_overlay(shape)

img2 = io.imread('red.jpg')
win2 = dlib.image_window()
win2.clear_overlay()
win2.set_image(img2)

dets2 = detector(img2, 1)
for k, d in enumerate(dets2):
    print("Detection {}: Left: {} Top: {} Right: {} Bottom: {}".format(
        k, d.left(), d.top(), d.right(), d.bottom()))

    shape = sp(img2, d)
    win2.clear_overlay()
    win2.add_overlay(d)
    win2.add_overlay(shape)

face_descriptor = facerec.compute_face_descriptor(img, shape)
```

```
face_descriptor2 = facerec.compute_face_descriptor(img2, shape)

result = distance.euclidean(face_descriptor, face_descriptor2)
if result < 0.6:
    print("Ponieważ odległość jest mniejsza niż 0,6, zdjęcie przedstawia tę
samą osobę. a = ", result)
else:
    print("Odległość jest większa niż 0,6, więc na zdjęciu nie ta sama
osoba. a = ", result)

input("Kliknij Enter!")
```

2. Dla zaawansowanych (na 5!):
   Wybierz osobę publiczną i porównaj jej zdjęcia dostępne w Internecie

Pliki graficzne ze stron internetowych można ręcznie umieścić do pewnego katalogu albo wczytać bezpośrednio za pomocą URL.

```
import face_recognition
import os
import cv2


KNOWN_FACES_DIR = 'known_faces'
UNKNOWN_FACES_DIR = 'unknown_faces'
TOLERANCE = 0.6
FRAME_THICKNESS = 3
FONT_THICKNESS = 2
MODEL = 'cnn'  # default: 'hog', other one can be 'cnn' - CUDA accelerated
(if available) deep-learning pretrained model


# Returns (R, G, B) from name
def name_to_color(name):
    # Take 3 first letters, tolower()
    # lowercased character ord() value rage is 97 to 122, substract 97,
multiply by 8
    color = [(ord(c.lower())-97)*8 for c in name[:3]]
    return color


print('Loading known faces...')
known_faces = []
known_names = []

# We organize known faces as subfolders of KNOWN_FACES_DIR
# Each subfolder's name becomes our label (name)
for name in os.listdir(KNOWN_FACES_DIR):

    # Next we load every file of faces of known person
    for filename in os.listdir(f'{KNOWN_FACES_DIR}/{name}'):

        # Load an image
        image =
face_recognition.load_image_file(f'{KNOWN_FACES_DIR}/{name}/{filename}')

        # Get 128-dimension face encoding
        # Always returns a list of found faces, for this purpose we take
first face only (assuming one face per image as you can't be twice on one
image)
```

```
        encoding = face_recognition.face_encodings(image)[0]

        # Append encodings and name
        known_faces.append(encoding)
        known_names.append(name)


print('Processing unknown faces...')
# Now let's loop over a folder of faces we want to label
for filename in os.listdir(UNKNOWN_FACES_DIR):

    # Load image
    print(f'Filename {filename}', end='')
    image =
face_recognition.load_image_file(f'{UNKNOWN_FACES_DIR}/{filename}')

    # This time we first grab face locations - we'll need them to draw
boxes
    locations = face_recognition.face_locations(image, model=MODEL)

    # Now since we know loctions, we can pass them to face_encodings as
second argument
    # Without that it will search for faces once again slowing down whole
process
    encodings = face_recognition.face_encodings(image, locations)

    # We passed our image through face_locations and face_encodings, so we
can modify it
    # First we need to convert it from RGB to BGR as we are going to work
with cv2
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    # But this time we assume that there might be more faces in an image -
we can find faces of dirrerent people
    print(f', found {len(encodings)} face(s)')
    for face_encoding, face_location in zip(encodings, locations):

        # We use compare_faces (but might use face_distance as well)
        # Returns array of True/False values in order of passed known_faces
        results = face_recognition.compare_faces(known_faces,
face_encoding, TOLERANCE)

        # Since order is being preserved, we check if any face was found
then grab index
        # then label (name) of first matching known face withing a
tolerance
        match = None
        if True in results:  # If at least one is true, get a name of first
of found labels
            match = known_names[results.index(True)]
            print(f' - {match} from {results}')

            # Each location contains positions in order: top, right,
bottom, left
            top_left = (face_location[3], face_location[0])
            bottom_right = (face_location[1], face_location[2])

            # Get color by name using our fancy function
            color = name_to_color(match)

            # Paint frame
```

```
            cv2.rectangle(image, top_left, bottom_right, color,
FRAME_THICKNESS)

            # Now we need smaller, filled grame below for a name
            # This time we use bottom in both corners - to start from
bottom and move 50 pixels down
            top_left = (face_location[3], face_location[2])
            bottom_right = (face_location[1], face_location[2] + 22)

            # Paint frame
            cv2.rectangle(image, top_left, bottom_right, color, cv2.FILLED)


# Wite a name
            cv2.putText(image, match, (face_location[3] + 10,
face_location[2] + 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (200, 200, 200),
FONT_THICKNESS)

    # Show image
    cv2.imshow(filename, image)
    cv2.waitKey(0)
    cv2.destroyWindow(filename)

input("Press Enter!")
```