计算机视觉 (实验二)
人脸检测与动态跟踪

智科三班　严中圣　222020335220177

2022 年 11 月 30 日

1　实验目的

图像处理 (image processing)，用计算机对图像进行分析，以达到所需结果的技术。图像处理将会是物联网产业发展的重要支柱之一。本实验旨在加深本课程中图像预处理、人脸检测、后处理等重要的知识点的理解和实践，并实现针对视频中的人脸检测和追逐算法，最终生成标注后的视频/流媒体。

2　实验环境

- Visual Studio Code 1.73.1
- OS: Windows 11 22H2
- CPU: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
- Packages: python 3.9.12, opencv 4.6.0.66, numpy 1.21.5, dlib 19.24.0

3　实验内容

(1) 通过笔记电脑摄像头进行逐帧的人脸检测，检测其中出现的人脸（可出现多个人脸）；

(2) 根据检测出的人脸，对其运动进行人脸追踪；

(3) 对检测结果进行后处理，将结果在图像中进行标注，并实时播放；

(4) 效率思考，若逐帧处理效率较低，考虑提高优化方法。

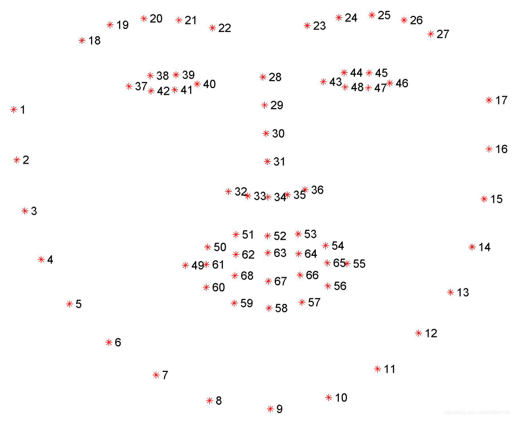
4　实验步骤

(1) Dlib 人脸特征检测与人脸数据库建立

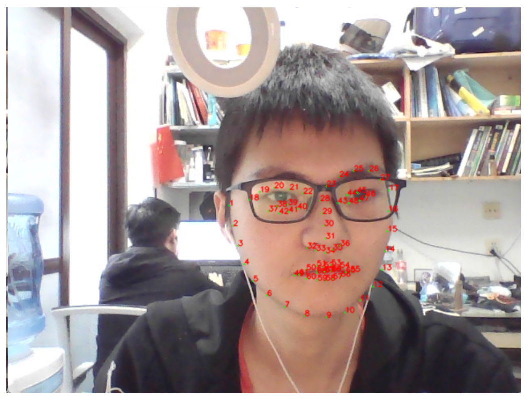
为了实现人脸识别的任务，我们首先对人脸图像进行了抓取并建立数据库保存对应人脸特征信息。我们在视频流或从网络抓取不同人脸特征，并保存为 480×480 大小的图片文件各 20 张，同时为了增加样本多样性，在采集的过程同时进行了亮度和对比度的随机调整，图像采集代码见附录 A.1。

图 1: 图像采集结果

在特征提取过程中，我们利用了 Dlib 库调用 68 特征点模型进行人脸特征提取，如图 3所示，代码见附录 2。根据抓取的图片和人脸识别模型训练得到的平均特征值存入 csv 文件保存为人脸数



(a) 人脸 68 特征点模型



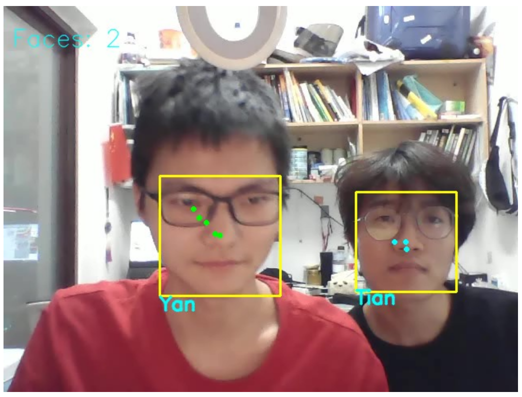
(b) 特征提取演示

图 2: 人脸特征采集过程

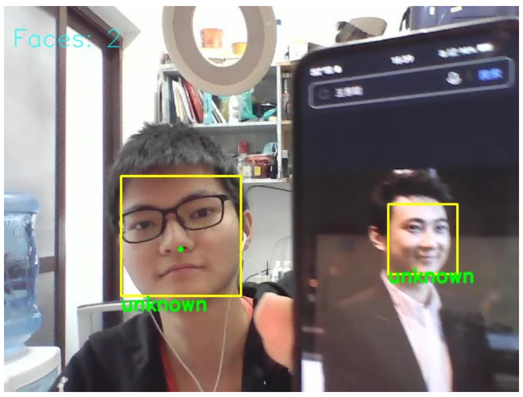
据库，方便后续人脸识别进行匹配。

(2) 人脸识别与动态轨迹跟踪

我们从视频流逐帧读入数据，同样首先利用 68 点人脸特征模型进行人脸识别，其次将检测到的人脸特征通过计算与数据集中保存特征的欧氏距离作对比来识别人脸，取欧氏距离最小的特征数据进行匹配。同时对不同识别人脸分别进行标记，对数据库中未录入的人脸认证为”Unknown”。同时记录人脸历史 15 帧的位置并逐帧绘制轨迹，即可实现动态跟踪。代码见附录 3，实现效果如下所示：



(a) 人脸识别效果展示 1



(b) 人脸识别效果展示 2

图 3: 人脸识别与动态跟踪过程。可对数据库中录入的人脸进行识别，同时检测到未录入的未知用户

A 附录

A.1 图像采集代码

```python
import cv2
import dlib
import os
import random

output_dir = 'C:/Users/11038/Desktop/ex2/faces/Yan'
size = 480

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# adjust the brightness and contrast of captured image
def relight(img, light=1, bias=0):
    w = img.shape[1]
    h = img.shape[0]
    for i in range(0, w):
        for j in range(0, h):
            for c in range(3):
                tmp = int(img[j, i, c]*light + bias)
                if tmp > 255:
                    tmp = 255
                elif tmp < 0:
                    tmp = 0
                img[j, i, c] = tmp
    return img


detector = dlib.get_frontal_face_detector()
camera = cv2.VideoCapture(0)
index = 1
```

```python
while True:
    if (index <= 20):
        print('Being processed picture %s' % index)
        success, img = camera.read()
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        dets = detector(gray_img, 1)

        for i, d in enumerate(dets):
            x1 = d.top() if d.top() > 0 else 0
            y1 = d.bottom() if d.bottom() > 0 else 0
            x2 = d.left() if d.left() > 0 else 0
            y2 = d.right() if d.right() > 0 else 0

            face = img[x1:y1, x2:y2]
            face = relight(face, random.uniform(0.5, 1.5), random.randint(-50,
                50))

            face = cv2.resize(face, (size, size))
            cv2.imwrite(output_dir+'/'+str(index)+'.jpg', face)
            index += 1
    else:
        camera.release()
        cv2.destroyAllWindows()
        break
```

A.2 特征提取代码

```python
# Features extraction from images and save into features_all.csv

import cv2
import os
import dlib
from skimage import io
import csv
import numpy as np

faces_dir_path = "C:/Users/11038/Desktop/ex2/faces/"

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
face_rec_model = dlib.face_recognition_model_v1("
    dlib_face_recognition_resnet_model_v1.dat")

# extract features of critical points for single image
def extract_128d_features(path_img):
    img_rd = io.imread(path_img)
    img_gray = cv2.cvtColor(img_rd, cv2.COLOR_BGR2RGB)
    faces = detector(img_gray, 1)

    if len(faces) != 0:
        shape = predictor(img_gray, faces[0])
        face_descriptor = face_rec_model.compute_face_descriptor(img_gray, shape
            )
    else:
        face_descriptor = 0
    return face_descriptor

# get mean features for single person
def get_mean_features_personX(path_faces_personX):
    features_list_personX = []
    photos_list = os.listdir(path_faces_personX)
    if photos_list:
        for i in range(len(photos_list)):
            features_128d = extract_128d_features(path_faces_personX + "/" +
                photos_list[i])
            if features_128d == 0:
                continue
            else:
                features_list_personX.append(features_128d)
    else:
        raise Exception("Warning: No images in {}".format(path_faces_personX))

    # N x 128D -> 1 x 128D
    if features_list_personX:
        features_mean_personX = np.array(features_list_personX).mean(axis=0)
    else:
        features_mean_personX = 0

    return features_mean_personX


people = os.listdir(faces_dir_path)
people.sort()

with open("C:/Users/11038/Desktop/ex2/features/faces_features_dataset.csv", "w",
     newline="") as csvfile:
    writer = csv.writer(csvfile)
    for person in people:
        print("##### " + person + " #####")
        # Get the mean/average features of face/personX, it will be a list with
            a length of 128D
        features_mean_personX = get_mean_features_personX(faces_dir_path +
            person)
        writer.writerow(features_mean_personX)
    print("Save all the features of {} faces.".format(len(people)))
```

A.3 实时检测与动态跟踪

```python
import os
import dlib
import time
import numpy as np
import cv2
import pandas as pd
from collections import deque

face_rec_model = dlib.face_recognition_model_v1(
    "dlib_face_recognition_resnet_model_v1.dat")


RGBs = [(255, 0, 0), (255, 255, 0), (0, 255, 0)]
font = cv2.FONT_HERSHEY_SIMPLEX

# compute the e-distance between two 128D features


def euclidean_distance(feature_1, feature_2):
    feature_1 = np.array(feature_1)
    feature_2 = np.array(feature_2)
    dist = np.sqrt(np.sum(np.square(feature_1 - feature_2)))
    return dist


faces_features_path = "C:/Users/11038/Desktop/ex2/features/
   faces_features_dataset.csv"
faces_features_csv = pd.read_csv(faces_features_path, header=None)

# the array to save the features of faces in the database
faces_features_dataset = []

for i in range(faces_features_csv.shape[0]):
    features_someone_arr = []
    for j in range(0, len(faces_features_csv.loc[i, :])):
        features_someone_arr.append(faces_features_csv.loc[i, :][j])
    faces_features_dataset.append(features_someone_arr)
print("Faces in Database:", len(faces_features_dataset))

center_points = []
for i in range(len(faces_features_dataset)):
    center_points.append(deque(maxlen=15))

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

fourcc = cv2.VideoWriter_fourcc(*'DIVX')
cap = cv2.VideoCapture(0)
out2 = cv2.VideoWriter('output.avi', fourcc, 30.0, (640,480), 1)

while cap.isOpened():
    flag, frame = cap.read()
    query = cv2.waitKey(1)
    img_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    faces = detector(img_gray, 0)  # get detected faces

    # the list to save the positions and names of current faces captured
    pos_namelist = []
    name_namelist = []

    # press 'q' to exit
    if query == ord('q'):
        break
    else:
        if len(faces) != 0:
            detected_faces_features = []
            for i in range(len(faces)):
                shape = predictor(frame, faces[i])  # feature extraction
                detected_faces_features.append(
                    face_rec_model.compute_face_descriptor(frame, shape))

            # traversal all the faces in the database
            for k in range(len(faces)):
                name_namelist.append("unknown")
                # the positions of faces captured
                pos_namelist.append(tuple([faces[k].left(), int(
                    faces[k].bottom() + (faces[k].bottom() - faces[k].top())/8)
                      ]))

                # for every faces detected, compare the faces in the database
                e_distance_list = []
                for i in range(len(faces_features_dataset)):
                    if str(faces_features_dataset[i][0]) != '0.0':
                        e_distance_tmp = euclidean_distance(
                            detected_faces_features[k], faces_features_dataset[i
                              ])
                        e_distance_list.append(e_distance_tmp)
                    else:
                        e_distance_list.append(255)

                # Find the one with minimum e distance
                similar_person_num = e_distance_list.index(
                    min(e_distance_list))

                if min(e_distance_list) < 0.4:
                    center_points[similar_person_num].append(
                        ((faces[k].left()+faces[k].right())/2, (faces[k].top()+faces[k].bottom())/2))
                    folder_name = 'C:/Users/11038/Desktop/ex2/faces/'
                    key_id = 0
                    names = os.listdir(folder_name)
                    for name in names:
                        if similar_person_num == key_id:
                            name_namelist[k] = name
                        key_id += 1

                    for i, d in enumerate(faces):
                        x1 = d.top() if d.top() > 0 else 0
                        y1 = d.bottom() if d.bottom() > 0 else 0
                        x2 = d.left() if d.left() > 0 else 0
                        y2 = d.right() if d.right() > 0 else 0
                        face = frame[x1:y1, x2:y2]
                        size = 256
                        face = cv2.resize(face, (size, size))
                        save_dir = "recordings/"
                        now_time = time.strftime(
                            "%Y-%m-%d-%H-%M-%S", time.localtime())
                        save_name = str(now_time)+str(name_namelist[k])+'.jpg'
                        save_path = save_dir + save_name
                        visitor_names = os.listdir(save_dir)
                        visitor_name = ''
                        for name in visitor_names:
                            # 名字切片到分钟数：2019-06-26-11-33-00yan.jpg
                            visitor_name = (name[0:16]+'-00'+name[18:])
                        visitor_save = (save_name[0:16]+'-00'+save_name[18:])
                        # 一分钟之内重复的人名不保存
                        if visitor_save != visitor_name:
                            cv2.imwrite(save_path, face)
                            print('Found people:' + save_dir +
                                  str(now_time) + str(name_namelist[k])+'.jpg')
                        else:
                            pass
                else:
                    for i, d in enumerate(faces):
                        x1 = d.top() if d.top() > 0 else 0
                        y1 = d.bottom() if d.bottom() > 0 else 0
                        x2 = d.left() if d.left() > 0 else 0
                        y2 = d.right() if d.right() > 0 else 0
                        face = frame[x1:y1, x2:y2]
                        size = 256
                        face = cv2.resize(face, (size, size))
                        path_visitors_save_dir = "recordings/"
                        now_time = time.strftime(
                            "%Y-%m-%d-%H-%M-%S", time.localtime())
                        save_path = path_visitors_save_dir + \
                            str(now_time)+'unknown.jpg'
                        cv2.imwrite(save_path, face)
                        print('Stranger:'+path_visitors_save_dir +
                              str(now_time)+'unknown.jpg')

                for kk, d in enumerate(faces):
                    cv2.rectangle(frame, tuple([d.left(), d.top()]), tuple(
                        [d.right(), d.bottom()]), (0, 255, 255), 2)
                    # print(center_points[similar_person_num])
                    for i in range(len(center_points[similar_person_num])):
                        cv2.circle(
                            frame, (int(center_points[similar_person_num][i][0])
                               , int(center_points[similar_person_num][i][1])),
                               2, RGBs[similar_person_num], 2)

            # write names under rectangle
            for i in range(len(faces)):
                cv2.putText(
                    frame, name_namelist[i], pos_namelist[i], font, 0.8, RGBs[
                       similar_person_num], 2, cv2.LINE_AA)

    print("Faces in camera now:", name_namelist)
    cv2.putText(frame, "Faces: " + str(len(faces)), (10, 50),
                font, 1, (255, 255, 0), 1, cv2.LINE_AA)
    cv2.imshow("camera", frame)
    out2.write(frame)

cap.release()
cv2.destroyAllWindows()
```