|  |  |  |  |
| --- | --- | --- | --- |
| 武汉大学国家网络安全学院实验报告 | | | |
| 课程名称 | 内容安全实验 | 实验日期 | 2024.5.9 |
| 实验名称 | 实验6：考勤系统 | | |
| 姓名 | 学号 | 专业 | 班级 |
| 周姝彤 | 2021302181061 | 网络空间安全 | 自强班 |
| 1. 实验目的及实验内容   （本次实验得具体内容；必要的原理分析） 1.1 实验任务 实现基于人脸识别和活体检测技术，实现基于BS架构的班级考勤系统。前端将摄像头采集的人脸信息传输到后端，后端进行活体检测、人脸库比对，返回人脸考勤信息到前端进行渲染显示。 1.2 实验要求  * BS架构的服务端采用Django、Flask、Python皆可，客户端采用VueJS、AngularJS、 HTML皆可 * 对接不少于2种活体检测方案（如Saas API、现有模型、自己实现），鼓励实现自己的活体检测方案，系统前端界面可切换方案 * 能抵抗照片/视频攻击 * 系统能正常运行，需现场检查验证  2 实验原理 根据具体的实验要求，在这里我选择了服务端采用FLASK，客户端使用HTML的实现方式。针对两个活体检测方案，包括本地活体检测（在不联网的情况下，也就是不搭配有API检测），联网活体检测（在联网情况下，搭配API检测和本地检测，从而加大检测的正确性） 2.1本地检测 采用开源框架dlib的shape\_predictor\_68\_face\_landmarks模型，对人脸的68个特征点进行检测定位。  本系统活体检测通过指定动作（包括左摇头、右摇头、眨眼、张嘴），故需要左眼[37,42]、右眼[43,48]、上嘴唇内边缘[66,68]等多个部分的特征点集合。 | | | |
| 2.1.1 眨眼检测   **检测原理：**眨眼检测的基本原理是计算眼睛长宽比EAR（Eye Aspect Ratio）值。当人眼睁开时，EAR在某个值上下波动。当人眼闭合时，EAR将迅速下降，理论上接近于零，实际上一般波动于0.25上下，故本系统设置阈值在0.25。眨眼速度比较快，一般1~3帧就完成了眨眼动作。本系统设置为2。    **EAR的计算公式如下：**    其中，p1~p5为当前眼睛的6个标记点。   2.1.2 张嘴检测 **检测原理：**类似眨眼检测，计算Mouth Aspect Ratio,MAR。当MAR大于设定的阈值时，认为张开了嘴巴。使用欧几里德距离来衡量嘴巴特征点之间的距离，并计算嘴巴张开程度的一个比率。    **对应的数学公式为：**  选择特征点：左右嘴角 (0, 6)，上下唇部 (1, 11), (2, 10), (3, 9), (4, 8), (5, 7)  欧几里德距离计算：  垂直距离：  A = dist.euclidean(mouth[1], mouth[11])  B = dist.euclidean(mouth[2], mouth[10])  C = dist.euclidean(mouth[3], mouth[9])  D = dist.euclidean(mouth[4], mouth[8])  E = dist.euclidean(mouth[5], mouth[7])  水平距离：  F = dist.euclidean(mouth[0], mouth[6])  计算嘴巴纵横比 (MAR)：  MAR=   2.1.3 左右脸转动检测 **检测原理：**通过计算左右脸特征点之间的欧几里德距离比率，来量化脸部的左右转动情况。  **对应的数学实现：**  ​ 特征点选择：特征点的选择基于 dlib 或类似库检测的 68 个面部特征点：  face[0] 和 face[16] 分别是左右脸颊的最外侧点。  face[27] 是鼻梁的上部中点。  face[30] 是鼻尖点。  face[48] 和 face[54] 分别是嘴巴两侧的点。  欧几里德距离计算：  右脸到中心点的距离：  rightA = dist.euclidean(face[0], face[27])：右脸颊到鼻梁上部中点的距离。  rightB = dist.euclidean(face[2], face[30])：右脸侧到鼻尖点的距离。  rightC = dist.euclidean(face[4], face[48])：右脸侧到嘴巴右侧点的距离。  左脸到中心点的距离：  leftA = dist.euclidean(face[16], face[27])：左脸颊到鼻梁上部中点的距离。  leftB = dist.euclidean(face[14], face[30])：左脸侧到鼻尖点的距离。  leftC = dist.euclidean(face[12], face[54])：左脸侧到嘴巴左侧点的距离。  计算左右脸比例 (FR)：  分别计算右脸到中心点距离与左脸到中心点距离的比率：        然后计算这三个比率的平均值，得到左右脸转动比例：    解释：  ​ 这些比率反映了面部左右两侧相对于中心线的对称程度：  如果脸部完全正对摄像头，rightA 和 leftA（以及 rightB 和 leftB、rightC 和 leftC）应当非常接近，导致比率接近 1。  如果脸部向右转动，右侧距离会变小，左侧距离会变大，导致比率小于 1。  如果脸部向左转动，右侧距离会变大，左侧距离会变小，导致比率大于 1。  通过计算这些比率并取平均值，可以得到一个整体的左右转动比例。  阈值设置：  在本方案中，我们将向右转动的阈值设置为0.5，向左转动的阈值设置为2。   2.2 API检测 在这里，我选用的是百度在线图片活体V3 检测。  **百度API的功能：**  人脸基础信息：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。  人脸质量检测：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。  基于图片的活体检测：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。  **活体检测存在几个标准的指标，如下所示：**  拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。  误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。  通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。  阈值（Threshold）：高于此数值，则可判断为活体。  **检测方法：**  针对图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。 2.3 人脸比对 将给定的人脸特征与已知的人脸特征进行比较，并返回最匹配的结果。np.linalg.norm 计算的是输入人脸特征向量和已知人脸特征向量之间的欧几里德距离。 2.4 算法创新点 1、 现有的一些实现是在人脸比对的过程中，比对的是数据库中的图片和要检测的图片，这样就导致了在检测过程中，需要对数据库中的图片进行特征提取，这就造成了时间的浪费，并且这一步每次检测都很重复，造成了算力的大量浪费。而在我的实现中，则是通过提前对图片进行特征提取，在比对过程中，直接将人脸特征与数据库中存储的特征进行比对，简化掉了人脸特征提取这一步骤，加快了算法的实行速度。  2、 现有的一些算法对于比对成功，有部分是当第一个检测欧几里得距离小于阈值距离，就直接返回检测的结果，这样虽然可以造成算法速度的提升，但是对应的准确率也会下降。在这里，我选择了比对所有的数据库中的特征向量，将小于设置阈值的数据存储起来，等到比对结束，选择其中距离最小的那个，这样就可以进一步保证我们识别的准确率。  这里我们的阈值设置为0.6。 | | | |
| 1. 实验步骤及结果分析   （详细描述实验步骤，并根据具体实验，记录、整理相应的数据表格等，对实验结果进行分析） | | | |
| 3.1 前端代码分析3.1.1 CSS样式 首先是对网页的一些风格设置。   3.1.2 HTML结构 <h1>: 标题。  <div class="main-container">: 包含视频、按钮和信息显示区的主容器。  <div id="videoContainer">: 包含视频元素、按钮和方法选择的容器。  <video id="videoElement" autoplay></video>: 用于显示摄像头的视频流。  <div class="button-group">: 按钮组，包含生成问题、开启摄像头、开始考勤和关闭摄像头按钮。  <select id="methodSelect">: 下拉菜单，用于选择活体检测方法（本地识别或联网识别）。  <div class="info-container">: 信息容器，包含问题详情、活体检测结果和考勤信息的显示区。  <div id="questionContainer">: 问题详情显示区，包含问题文本。  <div id="results">: 活体检测结果显示区，包含检测结果和人脸照片。  <div id="studentInfo">: 考勤信息显示区，包含考勤结果文本。   3.1.3 JavaScript 逻辑 **获取页面元素的引用:**    **生成问题按钮的点击事件:**  当用户点击 startButton 时，会向服务器发送请求生成一个随机问题，并将问题显示在页面上。  具体分析：  在点击按钮后，使用 fetch 函数向服务器端的 /generate\_question 端点发送一个 GET 请求。  当服务器返回响应时，使用 .then 方法处理该响应。并将响应转换为 JSON 格式。  转换为 JSON 格式的数据将传递给下一个 .then 方法的回调函数。回调函数接收 JSON 数据对象（data）后会进行将问题模块显示出来，并存储返回的问题，然后并在displayQuestion 元素上显示文本内容。    **开启摄像头按钮的点击事件:**  在用户点击“开启摄像头”按钮时，启动摄像头并将视频流显示在页面上的一个视频元素中。    **关闭摄像头按钮的点击事件:**  当用户点击关闭摄像头按钮时，function这个匿名函数会被调用。  首先检查 videoElement（一个 video 元素）的 srcObject 属性是否有值。如果摄像头已经被开启，这个属性会有一个视频流。  获取 srcObject 中的所有媒体轨道（音轨和视频轨道），然后对每个轨道调用 stop() 方法，停止轨道的捕获，并释放摄像头资源。    **开始考勤按钮的点击事件:**  当用户点击“开始考勤”按钮时，启动摄像头录制视频，然后将录制的视频发送到服务器进行处理，并显示处理结果，包括活体检测结果和识别到的面孔信息。  大致的流程如下：    **StartRecording函数：**  使用视频流stream初始化一个 MediaRecorder 对象，命名为 mediaRecorder。MediaRecorder 对象用于从媒体输入数据（如摄像头和麦克风）创建 Blob 对象，即录制视频。  设置事件监听器：  mediaRecorder.ondataavailable：当有数据可用时，会触发该事件。在这里，将可用数据块（即 Blob 对象）添加到 recordedChunks 数组中。这个数组用于存储录制的视频数据。  mediaRecorder.onstop：当录制停止时，会触发该事件。此处设置为调用 processVideo() 函数来处理录制后的视频。  开始录制：  调用 mediaRecorder.start() 方法开始录制视频。然后设置一个定时器，在10秒后调用 mediaRecorder.stop() 方法停止录制。这里录制的视频将持续10秒钟。    **ProcessVideo函数：**  创建 Blob 对象：  首先，将录制的视频数据 recordedChunks 转换为一个 Blob 对象，使用 Blob 构造函数，并指定 MIME 类型为 'video/webm'。  然后，创建一个 FormData 对象 formData。将录制的视频 Blob 对象以及其他数据（currentQuestion 和 method）附加到 FormData 对象中。    发起 POST 请求：  使用 fetch() 方法发起一个 POST 请求到服务器的 /process\_video 路径，将 FormData 对象作为请求的主体。    处理响应：  首先通过 .json() 方法将响应转换为 JSON 格式。（和问题生成部分原因一致）  如果识别到了人脸，将人脸信息显示在页面上，包括学生序号和活体检测结果。如果未识别到人脸，显示 "考勤失败"。将识别到的面孔图像显示在页面上。如果不存在，显示 "处理结果失败"。   3.2 后端代码分析3.2.1 App.py 路由 /generate\_question 处理生成问题请求：当客户端发起 /generate\_question 路径的 GET 请求时，服务器会随机选择一个问题，并将其作为 JSON 数据返回给客户端。  路由 /process\_video 处理处理视频请求：当客户端发起 /process\_video 路径的 POST 请求时，服务器会处理接收到的视频文件，并进行活体检测和人脸识别，最终返回处理结果。具体逻辑包括：  从 POST 请求中获取问题、视频文件和处理方法。  根据选择的处理方法进行活体检测，可以是本地检测或者网络检测。  如果活体检测通过，进行人脸识别，识别出视频中的人脸，并返回识别结果。  将识别到的人脸对应的图像转换成 Base64 编码，并将结果以 JSON 格式返回给客户端。  **generate\_question函数：**  定义了一个包含四个可能问题的列表 questions，包括 "blink eyes"、"open mouth"、"turn face right" 和 "turn face left"。使用 random.randint() 函数从问题列表中随机选择一个索引，从而得到一个随机问题的索引 index\_question，并返回给前端。    **process\_video函数：**  从前端传回来的数据中读取问题，方法和视频文件。    创建临时目录以保存上传的视频文件，并构建临时视频文件的路径。并将上传的视频文件保存到临时目录中。    使用 OpenCV 的 VideoCapture 对象打开临时视频文件。引入自定义的 main\_solve 类。    根据选择的处理方法（本地检测或网络检测），调用对应的活体检测方法，并根据结果更新 result\_message。    如果活体检测结果为真，则进行人脸识别。通过调用 recognize\_instant\_face 方法识别视频中的人脸，并将识别结果存储在 recognized\_faces 列表中。    如果有识别到的人脸，则获取其对应的图像路径，并将图像编码为 Base64 格式，存储在 recognized\_faces\_images 列表中。最后，将处理结果以 JSON 格式返回给客户端。    **具体的检测逻辑：**   3.2.2 Detect.py **关于本地检测：**  实现的流程为：  人脸检测：使用 self.detector 对象检测图像中的人脸，返回人脸矩形的列表 rects。  人脸特征提取：利用检测到的人脸矩形和面部特征点检测器，提取人脸的关键特征点（眼睛、嘴巴）。  特征点计算：根据提取的特征点，计算眼睛的长宽比（EAR）、嘴巴的张开程度（MAR）、左右脸转动的比例（FR）等。  凸包可视化：根据计算得到的凸包，将眼睛和嘴巴的区域用绿色轮廓可视化。  根据问题类型进行动作识别：根据问题的类型（blink eyes、open mouth、turn face right、turn face left），分别进行相应的动作识别和判断。    对应的特征点计算部分在前面的原理部分已经说明，下面我们关注于如何判别动作成功，以眨眼检测为例（其他动作的判别逻辑一致）：  如果是闭眼，则判断闭眼的连续帧数；当睁眼的时候，判断闭眼的连续帧数是否大于需要的连续帧数，如果满足，则判定存在眨眼。   3.2.3 关于BaiduAPI检测 在帧中读取数据帧，然后进行人脸检测，检测到就调用face\_api\_invoke函数。    **face\_api\_invoke函数：**  读取图片并编码为Base64格式：    依据官网的调用示例构建API请求:    发送API请求并处理响应:    处理API响应：  首先查看官网的返回示例，并关注验证方法：      Get\_access\_token函数：  依据官网示例写该函数。   3.2.4 关于人脸对比 recognize\_instant\_face函数实现从视频流中实时捕获帧并检测人脸，对检测到的人脸进行特征提取，将提取到的特征与已知人脸数据库中的特征进行比较，返回匹配到的已知人脸ID。    人脸对比的部分在前面的原理中也已经说明了。代码如下：   4运行效果： **页面展示：**    **问题生成：**    **摄像头开启：**    **考勤结果：**    **关闭摄像头：**    在下方按钮可以选择本地识别或者联网识别。 | | | |
| 1. 出现的问题及解决方法   （详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法） | | | |
| 报错：  Uncaught TypeError: Failed to execute 'append' on 'FormData': parameter 2 is not of type 'Blob'.  修改：  是因为 canvas.toBlob() 方法返回的不是一个有效的 Blob 对象。这可能是由于 canvas 元素的大小为零，因此无法正确地生成 Blob 对象。我们可以尝试在设置 canvas 大小之前确保视频元素已经加载完毕，以确保正确的尺寸。我们可以通过监听视频元素的 loadedmetadata 事件来做到这一点。  其余的都是FLASK框架搭建的问题，以及前后端视频传输，这些主要是因为我知识体系的不完备，所以这里就不赘述了 | | | |
| 1. 个人小结（描述实验心得，可提出实验的改进意见） | | | |
| 在本次实验中，我在第一种检测方法中实现了一个基于Dlib库的人脸识别与活体检测系统。该系统通过摄像头实时捕获视频流，并对视频中的人脸进行检测、特征提取和识别，同时实现了活体检测功能。主要功能包括实时人脸检测、特征提取、与已知人脸进行匹配以及活体检测。第二种方法我通过调用百度API来实现。  在实验中，系统能够检测到视频流中的人脸，并对人脸进行特征提取和识别。通过设定阈值（0.6）来判断特征匹配的准确性，实验结果表明该方法在识别已知人脸时具有较高的准确度。此外，活体检测功能通过面部动作检测和外部API验证，能够有效防止静态照片或视频的攻击。 | | | |
| 1. 教师评语及评分 | | | |
|  | | | |
| 教师签名：  年 月 日 | | | |