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	2024-5-13
实验名称	实验 6：考勤系统设计		
姓名	学号	专业	班级
邓鹏	2021302181152	网络空间安全	6 班

目录

一、实验目的及实验内容

1.1 实验目的

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

实验要求：
1、BS 架构的服务端采用 Django、Flask、Python 皆可，客户端采用 VueJS、AngularJS、 HTML 皆可
2、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
3、能抵抗照片/视频攻击
4、系统能正常运行，需现场检查验证

1.3 实验原理

1.3.1 人脸识别

人脸识别是一种基于生物特征的识别技术，通过分析和比较人脸的视觉特征来确认或辨认个体身份。随着计算机视觉和人工智能技术的发展，人脸识别技术已在安防监控、身份验证、智能设备等领域得到广泛应用。这里详细介绍人脸识别的基本原理及相关技术。

人脸识别的基本流程可以分为以下几个步骤：

- 人脸检测：从图像或视频中定位并提取人脸区域。
- 人脸对齐：将检测到的人脸图像进行标准化处理，使其在姿态、尺度和位置上尽可能一致。
- 人脸特征提取：从对齐后的人脸图像中提取具有区分性和鲁棒性的特征表示。
- 人脸匹配与识别：将提取到的特征与数据库中的特征进行比较，从而识别身份。

1. 人脸检测
人脸检测是人脸识别的第一步，旨在从输入的图像或视频中定位人脸区域。

常见的人脸检测方法包括：

• Haar 特征与 AdaBoost：Haar 特征是一种基于亮度变化的矩形特征，用于捕捉图像中的亮暗对比。AdaBoost 是一种机器学习算法，可以将多个弱分类器组合成一个强分类器。OpenCV 库中的 Haar 级联分类器便是基于此方法，通过层级分类器进行快速检测。

•HOG 特征与 SVM：方向梯度直方图（HOG）是一种描述局部梯度方向的特征。通过将图像划分为小块，并计算每个块内的梯度方向直方图，HOG 特征可以捕捉图像的边缘信息。支持向量机（SVM）是一种用于分类任务的监督学习算法，结合 HOG 特征，可以实现准确的人脸检测。

• 基于 CNN 的检测器：卷积神经网络（CNN）在图像分类和检测任务中表现优异。常用的基于 CNN 的人脸检测器包括 Faster R-CNN、YOLO 和 SSD 等。它们通过端到端的训练方式，能够有效地检测图像中的人脸区域。

• MTCNN：MTCNN 是一种专为人脸检测和对齐设计的多任务级联卷积网络。MTCNN 在检测人脸的同时，还能定位关键点，用于后续的对齐处理。其级联结构使得检测速度快、精度高，广泛应用于实际项目中。

2. 人脸对齐

人脸对齐的目的是将检测到的人脸图像标准化，使其在姿态、尺度和位置上尽可能一致。这一步骤非常重要，因为标准化的人脸图像可以提高后续特征提取和识别的准确性。

常见的人脸对齐方法包括：

• 眼睛对齐：通过检测人脸上的关键点（如眼睛、鼻子和嘴巴），将人脸图像旋转和平移，使两眼在水平线上。通过计算眼睛中心点之间的角度并进行旋转和缩放，可以使人脸图像在一定程度上标准化。

•Dlib 库中的 68 点检测器：Dlib 库提供了一种 68 点人脸关键点检测器。该方法通过回归树模型，在检测到人脸后，定位 68 个关键点（如眼角、嘴角、鼻尖等），从而实现人脸对齐。

3. 人脸特征提取

人脸特征提取是指从对齐后的人脸图像中提取具有区分性和鲁棒性的特征表示。这些特征可以用来区分不同个体。

常见的特征提取方法包括：

• LBP（局部二值模式）：LBP 是一种用于描述纹理特征的算子，通过比较像素与其周围像素的灰度值，将图像转换为二值模式，从而捕捉局部纹理信息。LBP 特征具有计算简单、鲁棒性强等优点，常用于早期的人脸识别系统。

• Fisherfaces 和 Eigenfaces：Fisherfaces 基于线性判别分析（LDA），通过最大化类间差异和最小化类内差异来提取特征。Eigenfaces 基于主成分分析（PCA），通过降维技术提取图像中的主成分特征。这两种方法提取的全局特征在一定程度上能够区分不同个体，但对光照和表情变化较为敏感。

• 深度卷积神经网络（CNN）：近年来，深度学习方法在特征提取方面表现出色。通过构建深度卷积神经网络，训练大量的人脸数据，可以学习到具有高度区分性的人脸特征。常见的网络结构包括 FaceNet、VGG-Face 和 ResNet 等。

• FaceNet：FaceNet 通过端到端的训练，将输入的人脸图像映射到一个 128 维的特征空间。该模型通过三重损失函数（Triplet Loss）进行训练，确保相同个体的特征向量距离尽可能近，不同个体的特征向量距离尽可能远。
• VGG-Face：VGG-Face 基于 VGG-16 网络结构，通过在大型人脸数据集上进行训练，提取深度特征。该模型在多个公开人脸数据集上表现优异，成为许多研究的基准模型。
• ResNet：ResNet 引入了残差连接，通过加深网络层数，提高模型的表达能力。在人脸识别任务中，ResNet 及其变体（如 ResNet-50、ResNet-101）广泛应用于特征提取。

4. 人脸匹配与识别
人脸匹配是指将提取到的特征与数据库中的特征进行比较，从而识别身份。

常用的匹配方法有：
• 欧氏距离：计算特征向量之间的欧氏距离，距离越小表明两张人脸越相似。该方法简单有效，广泛应用于实际系统中。
• 余弦相似度：计算特征向量之间的余弦相似度，值越大表明两张人脸越相似。该方法对特征向量的尺度不敏感，常用于人脸识别任务。
• 深度学习方法：通过训练二分类或多分类网络，直接输出匹配结果。例如，Siamese 网络通过学习相似度度量，在输入两张人脸图像后，输出其相似度分数，用于判断是否为同一人。

1.3.2 活体检测

活体检测技术是指用于区分真人和伪造物体（如照片、视频或 3D 模型）的一种生物识别技术。它在身份验证、安全监控等领域发挥着重要作用。随着伪造技术的不断发展，活体检测技术也在不断演进，以应对各种复杂的攻击手段。这里将详细介绍活体检测的基本原理及相关技术。

活体检测的目的是通过分析生物特征和行为特征，识别输入的人脸图像或视频是否来自于真人。常见的活体检测方法可以分为基于静态图像的方法、基于动态视频的方法和基于 3D 信息的方法。

1. 基于静态图像的检测方法：基于静态图像的检测方法主要依赖于分析图像的纹理、光照、反射等特征，以识别伪造物体。这些方法通常计算复杂度低，但在应对高质量伪造物体时，准确性可能有所不足。

• 纹理分析：纹理分析是基于图像的局部纹理特征来判断活体的有效方法。常见的纹理分析方法包括：
- LBP（局部二值模式）：通过比较像素与其周围像素的灰度值，将图像转换为二值模式，捕捉局部纹理信息。LBP 特征对光照变化具有较强的鲁棒性，但对高质量的伪造物体可能不足。
- Gabor 滤波器：通过一组具有不同频率和方向的 Gabor 滤波器提取图像的纹理特征。Gabor 特征可以有效地描述图像的局部纹理信息，但计算复杂度

较高。

• 频域分析：通过对图像进行傅里叶变换，将图像从时域转换到频域，分析图像的频域特征，可以有效地识别伪造图像的周期性纹理。例如，通过分析图像的频谱图，检测图像中的异常高频分量，可以识别伪造图像。

• 深度学习方法：深度学习方法在图像分类和识别任务中表现出色，通过训练卷积神经网络（CNN）来学习图像的局部和全局特征，可以实现高精度的静态活体检测。常见的方法包括：

- 基于 CNN 的分类器：通过在大规模的真实和伪造图像数据集上进行训练，CNN 可以学习到区分真实人脸和伪造人脸的特征表示。

- 生成对抗网络（GAN）：通过使用 GAN 生成伪造图像，并在训练过程中对抗训练，可以提高活体检测模型的鲁棒性。

2. 基于动态视频的检测方法

基于动态视频的检测方法通过分析视频帧序列中的时序信息，识别伪造视频的异常行为。这些方法通常能够应对动态伪造物体（如播放视频的人脸），具有较高的准确性。

• 基于光流的分析：光流法通过分析视频帧间的光流变化，检测视频中的运动模式。真实人脸在运动时，光流变化较为平滑，而伪造视频中的光流变化可能存在不自然的跳动和扭曲。通过计算视频帧间的光流场，可以识别伪造视频中的异常运动。

• 基于动作的检测：基于动作的检测方法通过要求用户完成特定动作（如眨眼、点头、张嘴等），通过检测这些动作的自然性来判断是否为真人。这些方法通常要求用户进行主动配合，能够有效地应对静态图片和视频伪造攻击。

• 基于深度学习的视频分析：使用循环神经网络（RNN）和长短时记忆网络（LSTM）等深度学习方法，分析视频帧序列中的时序特征，进行活体检测。通过在大量真实和伪造视频数据上进行训练，这些模型可以学习到区分真实视频和伪造视频的时序特征。

3. 基于 3D 信息的检测方法

基于 3D 信息的检测方法通过获取人脸的深度信息，识别伪造物体的平面特征。这些方法通常需要额外的硬件设备（如深度摄像头或结构光传感器），但能够提供更高的安全性。

• 结构光和深度摄像头：结构光通过投射特定的光栅图案，捕捉物体表面的 3D 结构信息，检测伪造人脸的平面特征。深度摄像头（如 Kinect）通过测量物体表面的深度信息，分析深度图像的真实性来进行活体检测。相比于普通摄像头，深度摄像头可以提供更丰富的 3D 信息，显著提高活体检测的准确性。

• 双目立体视觉：双目摄像头通过两个摄像头获取人脸的立体图像，计算视

差图，检测人脸的 3D 特征。通过分析双目图像的视差信息，可以区分真实人脸和平面伪造物体。这种方法的优点是无需复杂的硬件设备，缺点是计算复杂度较高。

1.3.3 人脸识别与活体检测结合的考勤打卡系统

我制作的考勤打卡系统结合了人脸识别和活体检测，包括以下几个步骤：
- 图像采集，通过摄像头获取员工的人脸图像或视频；
- 人脸检测，从采集的图像或视频中检测并提取人脸区域；
- 活体检测，对检测到的人脸区域进行活体检测，判断是否为真人；
- 人脸识别，通过人脸识别技术对通过活体检测的人脸进行身份验证；
- 考勤记录，将识别结果记录到考勤系统中，完成打卡过程。

1. 图像采集
图像采集是系统的起点，通过摄像头获取员工的人脸图像或视频。为了保证图像质量和检测准确率，通常需要考虑以下因素：摄像头的分辨率应足够高，以捕捉清晰的人脸图像；确保摄像头周围光照均匀，避免过暗或过亮的环境；尽量保证人脸正对摄像头，减少侧脸和其他复杂姿态。

2. 人脸检测
人脸检测是从图像或视频中定位并提取人脸区域的过程。常用的方法包括基于深度学习的卷积神经网络（CNN）和传统的 Haar 特征级联分类器等。现代人脸检测方法多采用深度学习技术，如 MTCNN 等。MTCNN 是一种多任务级联卷积网络，能够在不同的尺度上检测人脸，并同时定位人脸关键点。通过级联结构，MTCNN 可以在保持高准确率的同时，提高检测速度。

3. 活体检测
活体检测的目的是区分真人和伪造人脸，防止照片、视频等欺骗行为。常见的活体检测方法包括基于静态图像的检测、基于动态视频的检测和基于 3D 信息的检测。
基于静态图像的检测方法通过分析图像的纹理、光照、反射等特征，识别伪造物体。例如，使用局部二值模式（LBP）和深度卷积神经网络（CNN）进行分类。
基于动态视频的检测方法通过分析视频帧序列中的时序信息，识别伪造视频的异常行为。例如，基于光流法检测视频中的运动模式，或通过要求用户完成特定动作（如眨眼、点头等）来进行检测。基于 3D 信息的检测方法通过获取人脸的深度信息，识别伪造物体的平面特征。例如，使用结构光或深度摄像头获取人脸的 3D 结构信息。

4. 人脸识别
人脸识别是对通过活体检测的人脸进行身份验证的过程。常见的方法包括深度学习方法和特征匹配。深度学习方法使用深度卷积神经网络（CNN）提取人脸特征，并通过特征匹配进行识别。常用的模型有 FaceNet、VGG-Face 和 ResNet 等。特征匹配方法提取人脸图像的特征向量，并与数据库中的特征向量进行比较。常用的匹配方法包括欧氏距离和余弦相似度。

5. 考勤记录
考勤记录是将识别结果存储到考勤系统中的过程。系统需要记录员工的打卡时间、打卡地点和识别结果，并更新考勤数据库。通常还会提供日志和报表功能，方便管理员查看和管理考勤数据。

1.3.4 MobileNetV2 模型

MobileNetV2 是 Google 在 2018 年提出的一种轻量级卷积神经网络（CNN）模型，旨在提高移动和嵌入式设备上的深度学习应用性能。与其前身 MobileNetV1 相比，MobileNetV2 在结构上进行了多项改进，显著提高了模型的效率和准确性。

MobileNetV2 的结构可以概括为：
- 初始卷积层：提取低级特征。
- 倒残差模块：多个模块组成，每个模块包括扩展、深度卷积和压缩。
- 顶层卷积层：整合特征图。
- 全局平均池化层：将特征图压缩为单个值。
- 全连接层：用于分类任务。

1. 初始卷积层
在 MobileNetV2 的开头，有一个标准的卷积层，它用于处理输入图像，提取一些低级特征。就相当于打底，把图像的信息进行初步处理。

2. 倒残差结构
倒残差结构是 MobileNetV2 的核心创新之一。它是对传统残差结构的改进，主要步骤包括扩展、处理和压缩。
- 扩展：将输入特征图通过一个 1x1 的卷积层扩展为高维度特征图。就像是把一个简短的故事扩展成一个详细的小说，以便于更好地理解和处理。
- 处理：在扩展后的高维度特征图上进行深度卷积操作。这就像在扩展后的小说中进行逐字逐句的编辑和改进。这里的深度卷积是指每个通道独立处理，不混合通道间的信息。
- 压缩：通过另一个 1x1 的卷积层将高维度特征图压缩回原来的维度。就像把编辑好的详细小说重新缩写成一个简短的故事，但保留了所有重要的信息。

这样做的目的是提高特征表示的能力，同时保持计算效率。

3. 线性瓶颈层
MobileNetV1 中使用 ReLU 这样的激活函数来引入非线性，帮助模型学习复杂的特征。然而，在特征图压缩的过程中使用非线性激活函数可能会丢失重要的信息。所以 MobileNetV2 在压缩阶段引入了线性瓶颈层，不使用 ReLU 这样的非线性激活函数，以避免信息丢失。
可以理解为，在把小说缩写回简短故事时，不丢失关键信息，只做必要的简化。

4. 深度可卷积分离
与 MobileNetV1 相同，MobileNetV2 继续沿用了深度可卷积分离技术，深度可分

离卷积将标准卷积分解为两个步骤：
• 深度卷积：在输入的每个通道上分别进行卷积操作，不改变通道数。
• 逐点卷积：使用 1x1 卷积将深度卷积的输出组合起来，改变通道数。
这种分解方式相比标准卷积大大减少了计算量和参数量，同时保持了较高的特征提取能力。

5. 顶层卷积层和全局平均池化层
在网络的末尾，MobileNetV2 使用一个 1x1 的卷积层来进一步处理特征图。这一步相当于把前面所有提取的特征整合成一个最终的特征图。
之后，使用全局平均池化层将特征图的空间维度压缩到单个值。这就像是把一篇文章的所有内容总结成一个简洁的摘要。

6. 全连接层
最后，通过一个全连接层将压缩后的特征用于分类任务。这个层就像是根据摘要做出最终的判断，比如判断图片中是什么物体。

1.3.5 MiniFASNet 模型

MiniFASNet 模型是一个轻量级卷积神经网络，由 Minivision 开发，主要用于区分真实人脸和伪造人脸（例如纸张照片、屏幕图像、硅胶面具等）。

MiniFASNet 的网络结构包括多个卷积层、批量归一化层和 ReLU 激活函数。模型结构可以分为以下几个主要部分：
• 卷积层：用于提取输入图像的特征。通过多层卷积操作，逐步提取图像的低级和高级特征。
• 批量归一化层：用于加速训练过程，稳定模型。通过归一化输入的激活值，减小内部协变量偏移。
• ReLU 激活函数：引入非线性，提高模型的表达能力。ReLU 函数通过将所有负值设为零，引入非线性特性。
• SE 模块：增强特征选择能力。SE 模块通过自适应地重新校准特征图的通道，强调重要特征，抑制不重要特征。

1. SE 模块是 MiniFASNet 中的一个重要组件，它能够自适应地重新权衡通道间的关系。SE 模块通过以下几个步骤实现：
• Squeeze：对每一个特征通道进行全局平均池化，将特征图的空间信息压缩为通道间的全局描述。
•Excitation：通过一个两层的全连接网络自适应地学习通道间的相关性，并生成每个通道的权重。
• Recalibration：将生成的权重与原始特征图相乘，重新校准特征图的通道，突出重要特征。
通过 SE 模块，模型可以更加专注于重要特征，提高识别真实和伪造人脸的能力。

2. 多尺度模型融合
MiniFASNet 在训练过程中使用了多尺度模型融合方法，以提高模型的鲁棒性和

泛化能力。具体来说，它在不同尺度（大小）的图像块（patch）上进行训练。不同尺度的图像块捕捉了不同细节和上下文信息，使得模型能够在面对各种伪造人脸攻击时表现得更加稳健。通过在多尺度上训练模型，MiniFASNet 能够更好地捕捉伪造人脸的细微差别，从而提高防欺诈检测的准确性。

3. 傅里叶谱辅助监督
为了进一步提高模型的防欺诈检测能力，MiniFASNet 在训练过程中引入了傅里叶谱作为辅助监督信号。傅里叶谱可以反映图像在频域中的特征，真实人脸和伪造人脸在频域特征上存在明显差异。通过结合频域和空间域的信息，模型能够更全面地理解图像特征，从而提高识别真实和伪造人脸的能力。

4. 模型修剪
MiniFASNet 通过模型修剪技术，减少了计算量和参数量，同时尽量保持模型的精度。模型修剪主要包括以下几个步骤：
• 剪枝策略：根据一定的策略（如权重的大小、梯度等），选择需要剪掉的神经元或通道。
• 重新训练：在剪枝之后，重新训练模型，以恢复因剪枝而损失的部分性能。
• 迭代优化：多次进行剪枝和重新训练，直到达到预期的计算量和精度平衡。
模型修剪后的 MiniFASNet 具有更少的计算量和参数量，使其能够高效地运行在移动设备上。

1.3.6 MTCNN

MTCNN 是一种用于人脸检测和关键点定位的多任务级联卷积神经网络。它通过级联三个子网络（P-Net、R-Net 和 O-Net），逐步筛选并细化检测结果，实现高效且精确的人脸检测。以下是 MTCNN 的相关介绍。

1. 网络结构
MTCNN 由三个级联的子网络组成，每个子网络逐步提高检测的精度和准确度。
• P-Net（Proposal Network）
- 功能：生成初步的候选人脸框和五个关键点位置的估计。
- 结构：一个全卷积网络（Fully Convolutional Network），通过滑动窗口在不同尺度的图像上进行预测。
- 输出：候选框的坐标和置信度，关键点的位置。
- 处理：对输入图像进行多尺度检测，生成大量的候选框。使用非极大值抑制（Non-Maximum Suppression, NMS）来去除冗余候选框，保留高置信度的候选框。
• R-Net（Refinement Network）
- 功能：进一步精细化 P-Net 生成的候选框，去除错误检测并调整人脸位置。
- 结构：一个标准的卷积神经网络，输入为 P-Net 筛选后的候选框。
- 输出：更加精确的候选框和置信度。
- 处理：将 P-Net 生成的候选框输入 R-Net，过滤掉低置信度的候选框，输出更准确的人脸框。
• O-Net（Output Network）

- 功能：进一步精细化 R-Net 生成的候选框，并输出最终的人脸框和五个关键点位置。
- 结构：一个更深的卷积神经网络，输入为 R-Net 筛选后的候选框。
- 输出：最终精确的人脸框和关键点位置。
- 处理：对 R-Net 生成的候选框进行最后的筛选和校正，输出最终的人脸检测结果和关键点位置。

2. 工作流程

MTCNN 的工作流程可以分为以下几个步骤：

• 图像金字塔生成：为了处理不同尺度的人脸，生成输入图像的多尺度金字塔图像。

• P-Net 检测：对每个尺度的图像通过 P-Net 进行初步检测，生成候选人脸框及其置信度。对 P-Net 生成的候选框进行非极大值抑制（NMS），去除冗余候选框，保留高置信度的候选框。

• R-Net 精细化：将 P-Net 筛选后的候选框输入 R-Net 进行精细化处理。R-Net 对候选框进行进一步筛选和校正，过滤掉大部分错误检测，保留高置信度的候选框。

• O-Net 精细化：将 R-Net 筛选后的候选框输入 O-Net 进行最终的精细化处理。O-Net 进一步校正人脸框，并精确定位五个关键点（两眼、鼻子、两嘴角）。

• 关键点定位：O-Net 同时输出五个关键点的位置，包括两眼、鼻子和两嘴角的位置。

3. 非极大值抑制（NMS）

在 P-Net、R-Net 和 O-Net 的每个阶段，都会使用非极大值抑制（NMS）来去除冗余的候选框，确保保留的是置信度最高的候选框。

• 原理：NMS 通过比较重叠区域的置信度，去除重叠度较高且置信度较低的候选框。

• 步骤：
  - 根据置信度对候选框进行排序。
  - 依次选取置信度最高的候选框，去除与其重叠度超过阈值的其他候选框。
  - 重复上述步骤，直到所有候选框都处理完毕。

4. 多任务学习

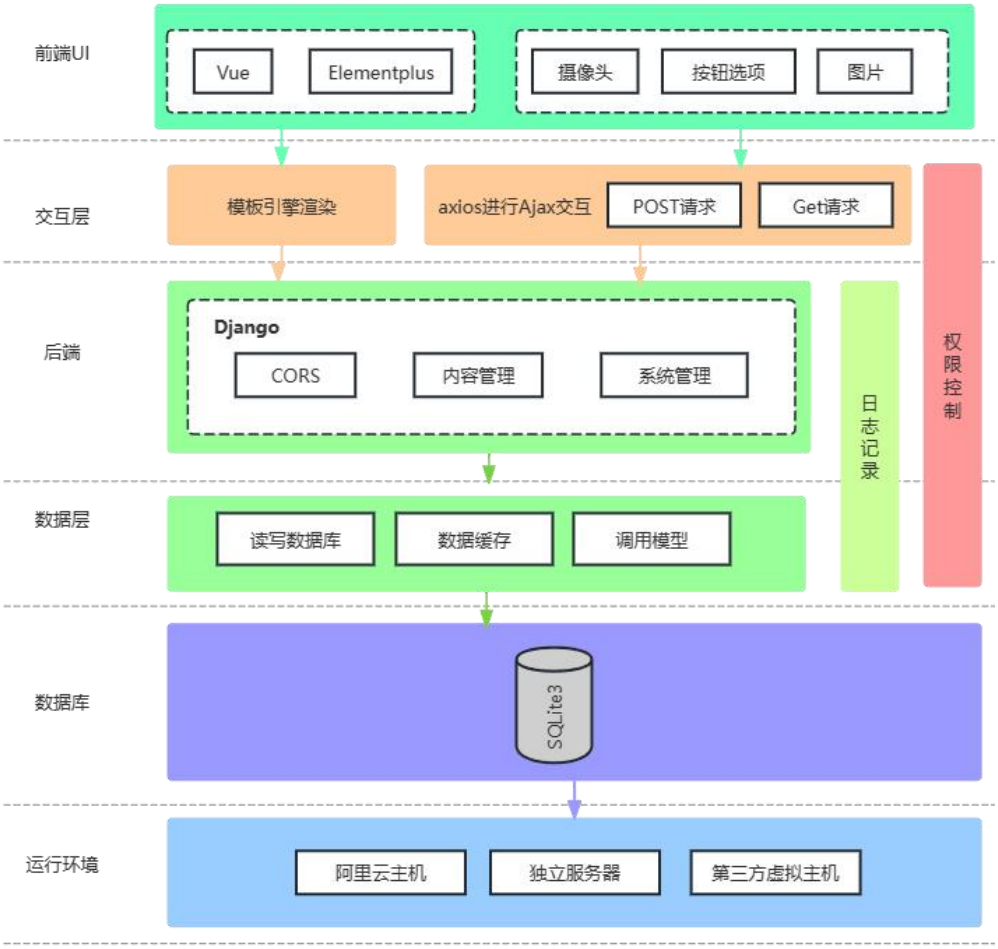
MTCNN 不仅进行人脸检测，还同时进行关键点定位，通过多任务学习提高了检测的准确度和鲁棒性。

人脸检测任务：通过分类器输出每个候选框的置信度，判断该区域是否包含人脸。
关键点定位任务：通过回归器输出每个候选框的关键点坐标，提高定位精度。

二、实验环境

Anaconda3 + Jupyter + torch + matplotlib + tensorflow + Vue + Django + Kaggle

系统架构如下



三、实验步骤及结果分析

首先，训练模型并跑通；然后前后端分离并调用模型。

3.1 在预训练的 MobileNetV2 模型上进行微调（自己实现）

使用 MobileNetV2 作为基础模型，进行了迁移学习，并作了若干调整来实现活体检测。

我选择了 MobileNetV2 作为基础模型，但不包含顶部的分类层。MobileNetV2 是一种高效的轻量级神经网络，适合在移动设备上运行。它在 ImageNet 上预训练过，已经学到了一些通用的特征，可以作为一个良好的特征提取器。通过去除顶部的分类层，可以添加自定义的分类层，使模型适应特定的任务。

```
# 使用预训练的MobileNetV2模型，这里不包含顶部的层，使用自定义的分类层
from tensorflow.keras.applications import MobileNetV2
pretrain_net = MobileNetV2(input_shape=(img_width, img_height, 3), include_top=False, weights=None)
```

为了提高模型的初始性能和加快收敛速度，我从预训练的 MobileNetV2 模型中加载了权重。这些权重是在 ImageNet 数据集上训练得到的，可以提供丰富的通用特征提取能力。迁移学习可以显著减少训练时间，并在数据量有限的情况下提高模型的表现。

```
# 加载预训练权重
load_param_path = '../input/mobilenet-v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5'
pretrain_net.load_weights(load_param_path)
```

在训练过程中，可以选择是否冻结部分层。冻结部分层的目的是保留预训练模型的特征提取能力，同时减少需要更新的参数数量。这有助于防止过拟合，特别是在数据量较小或训练资源有限的情况下。

```
# 冻结部分层
freeze_before = None

if freeze_before:
    for layer in pretrain_net.layers:
        if layer.name == freeze_before:
            break
        else:
            layer.trainable = False
```

在 MobileNetV2 的基础上，我添加了自定义的分类层。这些层包括一个卷积层（Conv2D）、一个 Dropout 层、一个全局平均池化层和一个全连接层作为最终的分类器。
具体设计如下：
- 卷积层（Conv2D）：进一步提取高层次特征。
- Dropout 层：在训练过程中随机丢弃部分神经元，防止过拟合。
- 全局平均池化层（Global Average Pooling）：将特征图的空间维度压缩到单个值，减少参数数量。
- 全连接层（Dense）：输出最终的分类结果。

In [10]:
```
# 添加自定义层
x = pretrain_net.output
x = Conv2D(32, (3, 3), activation='relu')(x)
x = Dropout(rate=0.2, name='extra_dropout1')(x)
x = GlobalAveragePooling2D()(x)
x = Dense(1, activation='sigmoid', name='classifier')(x)

model = Model(inputs=pretrain_net.input, outputs=x, name='mobilenetv2_spoof')
print(model.summary())
```

```
Model: "mobilenetv2_spoof"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)  0
_________________________________________________________________________________________________
Conv1_pad (ZeroPadding2D)    (None, 225, 225, 3)   0           input_1[0][0]
_________________________________________________________________________________________________
Conv1 (Conv2D)               (None, 112, 112, 32) 864         Conv1_pad[0][0]
_________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)  (None, 112, 112, 32) 128         Conv1[0][0]
_________________________________________________________________________________________________
Conv1_relu (ReLU)            (None, 112, 112, 32)  0           bn_Conv1[0][0]
_________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 112, 112, 32) 288         Conv1_relu[0][0]
_________________________________________________________________________________________________
expanded_conv_depthwise_BN (Bat (None, 112, 112, 32) 128         expanded_conv_depthwise[0][0]
_________________________________________________________________________________________________
expanded_conv_depthwise_relu (R (None, 112, 112, 32) 0           expanded_conv_depthwise_BN[0][0]
```

以上便是对预训练的 MobileNetV2 模型的模型结构修改部分。接下来是对数据的调整以及训练测试部分。

使用 ImageDataGenerator 对数据进行预处理和增强。训练数据进行了各种增强操作，如旋转、平移、剪切、缩放和水平翻转，以增加数据的多样性。验证和测试数据仅进行了归一化处理，以确保输入的一致性和模型的稳定性。

数据预处理与增强

In [6]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   rotation_range = 20,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.15,
                                   zoom_range = 0.15,
                                   horizontal_flip = True,
                                   fill_mode="nearest")

val_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale=1./255) if set_length["test"] > 0 else None
```

设置训练的参数，包括批量大小、学习率和训练轮数。同时，使用 Adam 优化器和二元交叉熵损失函数。Adam 优化器具有自适应学习率调整机制，适合处理稀疏梯度。二元交叉熵损失函数则适用于二分类问题。

训练参数

In [7]:
```
train_batch_size = 32
val_batch_size = 32
img_width = 224
img_height = 224

train_gen = train_datagen.flow_from_directory(train_dir,
                                              batch_size = train_batch_size,
                                              class_mode = 'binary',
                                              target_size = (img_width, img_height),
                                              seed = seed_number)

val_gen = val_datagen.flow_from_directory(val_dir,
                                          batch_size = val_batch_size,
                                          class_mode = 'binary',
                                          target_size = (img_width, img_height),
                                          seed = seed_number)

if test_datagen is not None:
  test_gen = test_datagen.flow_from_directory(test_dir,
                                              batch_size = 1,
                                              class_mode = 'binary',
                                              target_size = (img_width, img_height),
                                              seed = seed_number,
                                              shuffle=False)
else:
  test_gen = None
```

```
Found 8299 images belonging to 2 classes.
Found 2948 images belonging to 2 classes.
Found 7580 images belonging to 2 classes.
```

设置一些参数后，首先编译模型，使用 Adam 优化器和二元交叉熵损失函数。

Adam 优化器因其自适应学习率调整机制，非常适合处理稀疏梯度问题。二元交叉熵损失函数适用于二分类任务。然后，设置多个回调函数来辅助训练过程，如 ModelCheckpoint 用于在每个训练轮次结束时保存模型；TensorBoard 记录训练日志，并可以在 TensorBoard 中进行可视化。此外，由于数据集可能存在类别不平衡的问题（真实人脸和伪造人脸数量不平衡），我计算了类别权重。在训练过程中给予少数类别更多的关注，可以提高模型对少数类别的识别能力，从而提升整体性能。最后，使用预处理后的数据开始训练模型。训练过程中，模型的权重根据数据和损失函数进行更新。设置验证集可以实时监控模型在未见过的数据上的表现，从而评估其泛化能力。训练过程中，每个训练轮次的训练和验证准确率及损失都会被记录下来，以便后续分析。

训练

In [11]:
```
train_id = "lcc-train04b-weight_all"
num_epochs = 15
learning_rate = 5e-5

print(f"Training config of '{train_id}'...")
print(f"Number of epoch\t: {num_epochs}")
print(f"Initial LR\t: {learning_rate}")

# 编译模型
model.compile(optimizer = Adam(lr=learning_rate),
              loss = 'binary_crossentropy',
              metrics = ['acc'])

# 保存模型
save_dir = os.path.join("./", train_id)
if not os.path.isdir(save_dir):
  os.makedirs(save_dir)

cont_filepath = "mobilenetv2-epoch_{epoch:02d}.hdf5"
cont_checkpoint = ModelCheckpoint(os.path.join(save_dir, cont_filepath))

best_filepath = "mobilenetv2-best.hdf5"
best_checkpoint = ModelCheckpoint(os.path.join(save_dir, best_filepath),
                                  save_best_only=True,
                                  save_weights_only=True)

log_dir = os.path.join(save_dir, "logs")
use_tensorboard = TensorBoard(log_dir=log_dir,
                              histogram_freq=1,
                              update_freq="batch")

# 学习率
plateau_scheduler = ReduceLROnPlateau(factor=0.2, patience=3, verbose=1,
                                      min_delta= 0.005, min_lr=5e-7)


# 计算类别权重
train_length = len(train_gen.classes)
weight0 = train_length / case_count_df['train'][label_name[0]] * (1 / len(label_name))
weight1 = train_length / case_count_df['train'][label_name[1]] * (1 / len(label_name))
class_weight = {0: weight0, 1: weight1}

print(f"Class weight\t: {class_weight}")
```

```
Training config of 'lcc-train04b-weight_all'...
Number of epoch : 15
Initial LR      : 5e-05
Class weight    : {0: 3.392886345053148, 1: 0.5864188807235726}
```

开始训练

开始训练

In [12]:
```
history = model.fit(train_gen,
                    epochs = num_epochs,
                    steps_per_epoch = set_length['train'] // train_batch_size,
                    validation_data = val_gen,
                    validation_steps = 1,
                    callbacks = [best_checkpoint,
                                 cont_checkpoint,
                                 plateau_scheduler],
                    class_weight=class_weight)

history_df = pd.DataFrame.from_dict(history.history)
history_df.to_csv(os.path.join(save_dir, "history.csv"), index=False)
```

由于我的电脑性能较差，训练时间较长，我使用了 kaggle 中的 gpu 来训练模型，下面是具体细节。

```
Epoch 1/15
259/259 [==============================] - 276s 1s/step - loss: 0.2262 - acc: 0.8968 - val_loss: 0.3541 - val_acc: 0.9375 - lr: 5.0000e
-05
Epoch 2/15
259/259 [==============================] - 204s 789ms/step - loss: 0.0795 - acc: 0.9701 - val_loss: 0.8872 - val_acc: 0.9062 - lr: 5.00
00e-05
Epoch 3/15
259/259 [==============================] - 205s 790ms/step - loss: 0.0370 - acc: 0.9855 - val_loss: 0.2461 - val_acc: 0.9688 - lr: 5.00
00e-05
Epoch 4/15
259/259 [==============================] - 214s 826ms/step - loss: 0.0354 - acc: 0.9879 - val_loss: 0.1370 - val_acc: 0.9688 - lr: 5.00
00e-05
Epoch 5/15
259/259 [==============================] - 205s 792ms/step - loss: 0.0236 - acc: 0.9913 - val_loss: 0.4750 - val_acc: 0.8750 - lr: 5.00
00e-05
Epoch 6/15
259/259 [==============================] - 199s 769ms/step - loss: 0.0211 - acc: 0.9919 - val_loss: 0.2462 - val_acc: 0.8750 - lr: 5.00
00e-05
Epoch 7/15
259/259 [==============================] - ETA: 0s - loss: 0.0168 - acc: 0.9952
Epoch 00007: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
259/259 [==============================] - 197s 762ms/step - loss: 0.0168 - acc: 0.9952 - val_loss: 0.4042 - val_acc: 0.9375 - lr: 5.00
00e-05

    Epoch 8/15
    259/259 [==============================] - 198s 763ms/step - loss: 0.0150 - acc: 0.9944 - val_loss: 0.1556 - val_acc: 0.9688 - lr: 1.00
    00e-05
    Epoch 9/15
    259/259 [==============================] - 199s 767ms/step - loss: 0.0082 - acc: 0.9967 - val_loss: 0.0268 - val_acc: 1.0000 - lr: 1.00
    00e-05
    Epoch 10/15
    259/259 [==============================] - 206s 795ms/step - loss: 0.0067 - acc: 0.9983 - val_loss: 0.0959 - val_acc: 0.9688 - lr: 1.00
    00e-05
    Epoch 11/15
    259/259 [==============================] - 204s 787ms/step - loss: 0.0054 - acc: 0.9979 - val_loss: 0.0810 - val_acc: 0.9375 - lr: 1.00
    00e-05
    Epoch 12/15
    259/259 [==============================] - 206s 797ms/step - loss: 0.0076 - acc: 0.9973 - val_loss: 0.0016 - val_acc: 1.0000 - lr: 1.00
    00e-05
    Epoch 13/15
    259/259 [==============================] - 199s 769ms/step - loss: 0.0040 - acc: 0.9985 - val_loss: 0.0013 - val_acc: 1.0000 - lr: 1.00
    00e-05
    Epoch 14/15
    259/259 [==============================] - 197s 762ms/step - loss: 0.0055 - acc: 0.9976 - val_loss: 0.2975 - val_acc: 0.9688 - lr: 1.00
    00e-05
    Epoch 15/15
    259/259 [==============================] - ETA: 0s - loss: 0.0077 - acc: 0.9979
    Epoch 00015: ReduceLROnPlateau reducing learning rate to 1.9999999494757505e-06.
    259/259 [==============================] - 202s 780ms/step - loss: 0.0077 - acc: 0.9979 - val_loss: 0.0540 - val_acc: 0.9688 - lr: 1.00
    00e-05
```

之后是，训练结果的可视化，可见效果很好，准确率高达 99.79%。

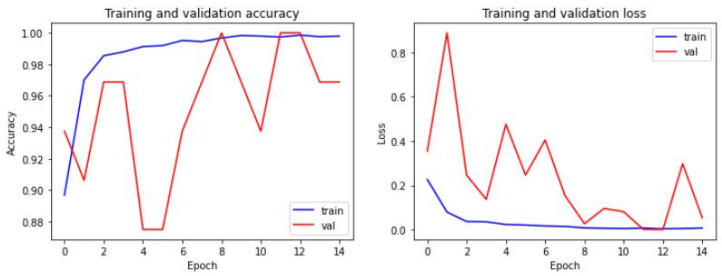
In [13]:
```
train_accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(train_accuracy))
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(epochs, train_accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['train', 'val'], loc='lower right')

plt.subplot(1,2,2)
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['train', 'val'], loc='upper right')

plt.show()
```



最后，使用测试集进行模型性能评估。计算准确率和损失；预测标签；绘制混淆矩阵；计算精确率、召回率和 F1 得分。

测试集性能评估

In [14]:
```
test_scores = model.evaluate(test_gen, steps=set_length['test'])
print("Test results Accuracy: {0:.2f}% and Loss: {0:.2f}".format(test_scores[1]*100, test_scores[0]))

threshold = 0.5
y_pred_value = np.squeeze(model.predict(test_gen, steps=set_length['test'], verbose=1))

y_pred = np.zeros(y_pred_value.shape).astype(np.int32)
y_pred[y_pred_value > threshold] = 1

y_true = test_gen.classes

print(f"Label\t\t: {y_true[:10]}")
print(f"Prediction\t: {y_pred[:10]}")
```

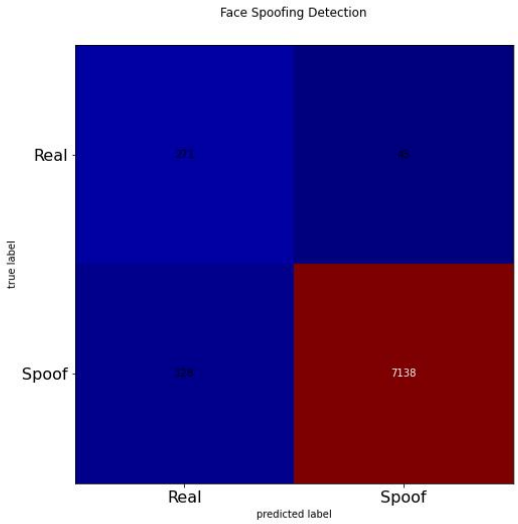
```
7580/7580 [==============================] - 180s 24ms/step - loss: 0.1016 - acc: 0.9774
Test results Accuracy: 97.74% and Loss: 97.74
7580/7580 [==============================] - 115s 15ms/step
Label           : [0 0 0 0 0 0 0 0 0 0]
Prediction      : [0 0 0 0 0 1 0 0 0 0]
```

In [15]:
```
# 绘制混淆矩阵
confusion_matrix_result = confusion_matrix(y_true, y_pred)
plot_confusion_matrix(confusion_matrix_result,
                      figsize=(12,8),
                      hide_ticks=True,
                      cmap=plt.cm.jet)
plt.title("Face Spoofing Detection")
plt.xticks(range(2), ['Real', 'Spoof'], fontsize=16)
plt.yticks(range(2), ['Real', 'Spoof'], fontsize=16)
plt.show()

tn, fp, fn, tp = confusion_matrix_result.ravel()
precision = tp / (tp+fp)
recall = tp / (tp+fn)
f1_score = 2 * precision * recall / (precision+recall)

print("Report Summary:")
print("Precision\t: {:.2f}%".format(precision*100))
print("Recall\t\t: {:.2f}%".format(recall*100))
print("F1 Score\t: {:.2f}%".format(f1_score*100))

print("\nNotes: ")
print("True labels\t: Spoof")
print("False labels\t: Real")
```




```
Report Summary:
Precision       : 99.40%
Recall          : 98.24%
F1 Score        : 98.82%

Notes:
True labels     : Spoof
False labels    : Real
```

3.2 MiniFASNet 模型的调用（Github）

模型是使用的 github 中找到的模型，下面直接调用。

首先初始化防欺诈检测器和图像裁剪器。使用 AntiSpoofPredict 类初始化防欺诈

检测模型，使用 CropImage 类初始化图像裁剪器。然后，将输入图像调整为 3:4 的比例，并调用 check_image 函数进行检查。如果图像比例不正确，检测过程将被终止。

```
def test(image, model_dir, device_id):
    model_test = AntiSpoofPredict(device_id)
    image_cropper = CropImage()
    # image = cv2.imread(SAMPLE_IMAGE_PATH + image_name)
    image = cv2.resize(image, (int(image.shape[0] * 3 / 4), image.shape[0]))
    result = check_image(image)
    if result is False:
        return
```

接下来，使用防欺诈检测器获取输入图像中的人脸边界框。这一步通过调用防欺诈检测器的 get_bbox 方法实现。然后，初始化一个空的预测结果数组 prediction，用于累加多个模型的预测结果，同时初始化预测速度计数器 test_speed，用于记录预测所花费的时间。

```
    image_bbox = model_test.get_bbox(image)
    prediction = np.zeros((1, 3))
    test_speed = 0
```

在处理每个模型时，遍历模型目录中的所有模型文件，解析每个模型的名称以获取输入尺寸、模型类型和缩放比例。根据解析结果设置裁剪参数，并使用图像裁剪器裁剪图像。调用防欺诈检测器的 predict 方法对裁剪后的图像进行预测，累加每个模型的预测结果和预测时间。最终，根据累加的预测结果，确定并返回最终的标签（0 表示真实人脸，1 表示伪造人脸）。

```
    for model_name in os.listdir(model_dir):
        h_input, w_input, model_type, scale = parse_model_name(model_name)
        param = {
            "org_img": image,
            "bbox": image_bbox,
            "scale": scale,
            "out_w": w_input,
            "out_h": h_input,
            "crop": True,
        }
        if scale is None:
            param["crop"] = False
        img = image_cropper.crop(**param)
        start = time.time()
        prediction += model_test.predict(img, os.path.join(model_dir, model_name))
        test_speed += time.time()-start

    # draw result of prediction
    label = np.argmax(prediction)
    value = prediction[0][label]/2

    return label
```

3.3 基于 MobileNet-0.5 与 MTCNN 的模型调用（Github）

利用 MTCNN 进行人脸检测，并使用预训练的深度学习模型进行活体检测，判

断输入的人脸图像是真实人脸还是假脸。

加载两个模型，一个是基于 MTCNN 的模型，用于人脸检测；另一个是预训练的深度学习模型，用于人脸活体检测。MTCNN 模型用于从图像中检测人脸并获取人脸的边界框和关键点。通过调用 load_mtcnn_model 函数加载 MTCNN 模型。而深度学习模型用于判断检测到的人脸是真实人脸还是假脸。通过 Keras 的 load_model 函数加载预训练的模型文件 fas.h5。

```
from mtcnn import MTCNN

from keras.models import load_model

model = load_model("../model/fas.h5")


1个用法
def load_mtcnn_model(model_path):
    mtcnn = MTCNN(model_path)
    return mtcnn
```

实现加载MTCNN模型和单张图像的活体检测，定义了几个辅助函数。test_one(X)函数用于对单张人脸图像进行活体检测，输入参数为人脸图像 X，处理过程包括将输入图像调整为 224x224 像素并进行归一化处理（减去 127.5 并除以 127.5），然后调用预训练的深度学习模型进行预测，输出为模型的预测结果，即图像为真实人脸的概率。

```
def test_one(X):
    TEMP = X.copy()
    X = (cv2.resize(X,(224,224))-127.5)/127.5
    t = model.predict(np.array([X]))[0]
    time_end=time.time()
    return t
```

test_camera 函数通过摄像头实时检测人脸，并进行活体检测。该函数的具体处理流程如下：首先使用 OpenCV 的 VideoCapture 打开摄像头设备，然后进入一个循环，持续从摄像头读取帧图像。如果读取成功，则使用 MTCNN 模型检测帧中的人脸，获取人脸的边界框和关键点。对于每个检测到的人脸，裁剪出人脸区域，并调用 test_one 函数进行活体检测。根据活体检测结果，在原始帧图像上绘制不同颜色的矩形框（绿色表示真实人脸，红色表示假脸）。最后，显示处理后的帧图像，并检查用户输入，如果按下'q'键，则退出循环并释放摄像头资源。

```
def test_camera(mtcnn,index = 0):

    cam=cv2.VideoCapture(0)

    while(1):
        success, frame = cam.read()
        if success:
            image = frame

            img_size = np.asarray(image.shape)[0:2]

            bounding_boxes, scores, landmarks = mtcnn.detect(image)
```

```
        nrof_faces = bounding_boxes.shape[0]
        if nrof_faces > 0:
            for det, pts in zip(bounding_boxes, landmarks):

                det = det.astype('int32')
                #print("face confidence: %2.3f" % confidence)
                det = np.squeeze(det)
                y1 = int(np.maximum(det[0], 0))
                x1 = int(np.maximum(det[1], 0))
                y2 = int(np.minimum(det[2], img_size[1]-1))
                x2 = int(np.minimum(det[3], img_size[0]-1))

                w = x2-x1
                h = y2-y1
                _r = int(max(w,h)*0.6)
                cx,cy = (x1+x2)//2, (y1+y2)//2

                x1 = cx - _r
                y1 = cy - _r

                x1 = int(max(x1,0))
                y1 = int(max(y1,0))

                x2 = cx + _r
                y2 = cy + _r

                h,w,c =frame.shape
                x2 = int(min(x2 ,w-2))
                y2 = int(min(y2, h-2))

                _frame = frame[y1:y2 , x1:x2]
                score = test_one(_frame)

                print(score)
                if score > 0.95:
                    cv2.rectangle(frame, (x1,y1) ,(x2,y2) , (0,255,0)  ,2)
                else:
                    cv2.rectangle(frame, (x1,y1) ,(x2,y2) , (255,0,0)  ,2)

        cv2.imshow("image", image)
        if cv2.waitKey(1) & 0XFF == ord('q'):
            cv2.destroyAllWindows()
            cam.release()
            break

    else:
        cv2.destroyAllWindows()
        cam.release()
        break
```

3.4 前后端代码编写

前端使用 Vue + Elementplus 编写，后端则使用 Django 编写。

3.4.1 前端代码编写

前端主页分为三个页面：人脸实时考勤、签到记录以及添加用户。

在添加用户模块中，首先使用 el-form 组件创建一个表单，表单包含用户姓名和学号的输入框（el-input 组件），以及一个文件上传控件用于上传用户照片。用户可以通过表单输入自己的姓名和学号，并上传一张个人照片。提交按钮由

el-button 组件实现，用于提交表单数据。表单数据和照片文件在脚本部分通过 form 对象和 selectedImage 变量进行管理。handleFileChange 方法用于处理文件上传事件，将选择的文件存储到 selectedImage 变量中。addVisitor 方法负责验证表单数据的完整性，并将数据和照片文件组成 FormData 对象，发送 POST 请求到服务器以添加用户。如果添加成功，清空表单并显示成功消息；否则，显示错误消息。

```
<template>
  <div>
    <el-form :model="form" label-width="120px">
      <el-form-item label="姓名">
        <el-input v-model="form.name"></el-input>
      </el-form-item>

      <el-form-item label="学号">
        <el-input v-model="form.student_id"></el-input>
      </el-form-item>

      <el-form-item label="上传自己的图片">
        <input type="file" @change="handleFileChange" accept="image/*">
      </el-form-item>

      <el-button type="success" @click="addVisitor">添加用户</el-button>
    </el-form>
  </div>
</template>

<script>
export default {
  data() {
    return {
      form: {
        name: '',
        student_id: '',
      },
      selectedImage: null,
    };
  },
  methods: {
    handleFileChange(event) {
      const file = event.target.files[0];
      if (file) {
        this.selectedImage = file;
      } else {
        this.selectedImage = null;
      }
    },
    async addVisitor() {
      if (!this.form.name || !this.form.student_id || !this.selectedImage) {
        this.$message.error('信息表单不完整!');
        return;
      }

      try {
        const formData = new FormData();
        formData.append('name', this.form.name);
        formData.append('student_id', this.form.student_id);
        formData.append('face_image', this.selectedImage);
        const response = await this.$axios.post('/visitors/', formData);
        console.log('添加用户:', response.data);
        this.$message.success('成功添加用户!');
        this.form = {
          name: '',
          student_id: '',
        };
        this.selectedImage = null;

      } catch (error) {
        console.error('添加用户失败:', error);
        this.$message.error('添加用户失败');
      }
    },
  },
};
</script>
```

在实时考勤模块中，使用 video 标签显示实时摄像头画面，使用 canvas 标签截取图像。用户可以输入学号，选择模型，并通过按钮截取实时图片、上传图片或重新获取图片。该模块提供了实时人脸识别和验证的功能。脚本部分定义了多个变量来管理摄像头流、截取的图片、输入的学号和选择的模型等。startCamera 方法负责获取摄像头权限并启动视频流，captureImage 方法用于从视频流中截取当前帧并将其转换为 Base64 格式。uploadImage 方法将截取的图片和学号、模型信息发送到服务器进行验证，并显示验证结果。resetImage 方法用于重置截取的图片和消息。

```
<template>
  <div>
    <video ref="video" width="640" height="480" autoplay></video>
    <canvas ref="canvas" width="640" height="480" style="display: none;"></canvas>
    <div v-if="capturedImage">
      <img :src="capturedImage" alt="Captured Image" width="640" height="480">
    </div>

    <!-- 模型选择和操作按钮 -->
    <div>
      <el-input v-model="studentId" placeholder="输入学号"></el-input>
      <el-select v-model="selectedModel" placeholder="选择模型">
        <el-option label="Model 1 (Default)" value="moedl1"></el-option>
        <el-option label="Model 2 (Alternative)" value="model2"></el-option>
        <el-option label="Model 3 (Alternative)" value="model3"></el-option>
        <!-- 可以添加更多模型 -->
      </el-select>
      <el-button @click="captureImage">截取实时图片</el-button>
      <el-button @click="uploadImage" type="primary" v-if="capturedImage">上传图片</el-button>
      <el-button @click="resetImage" v-if="capturedImage">重新获取图片</el-button>
    </div>

    <!-- 显示结果或消息 -->
    <div v-if="message">
      <el-alert :title="message" type="success" v-if="isSuccess" show-icon></el-alert>
      <el-alert :title="message" type="error" v-else show-icon></el-alert>
    </div>

    <!-- 显示已验证的访客信息 -->
    <div v-if="visitorInfo">
      <h3>Visitor Information:</h3>
      <p><strong>学号:</strong> {{ visitorInfo.student_id }}</p>
      <p><strong>姓名:</strong> {{ visitorInfo.name }}</p>
    </div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      videoStream: null,
      capturedImage: null,
      studentId: '',
      selectedModel: 'model1',
      message: '',
      isSuccess: false,
      visitorInfo: null
    };
  },
  mounted() {
    this.startCamera();
  },
  beforeDestroy() {
    if (this.videoStream) {
      this.videoStream.getTracks().forEach(track => track.stop());
    }
  },
```

```
    methods: {
      async startCamera() {
        try {
          this.videoStream = await navigator.mediaDevices.getUserMedia({ video: true });
          this.$refs.video.srcObject = this.videoStream;
        } catch (error) {
          console.error('无法获取摄像头权限:', error);
        }
      },
      captureImage() {
        const video = this.$refs.video;
        const canvas = this.$refs.canvas;
        const context = canvas.getContext('2d');
        context.drawImage(video, 0, 0, canvas.width, canvas.height);
        this.capturedImage = canvas.toDataURL('image/png');
      },
      async uploadImage() {
        if (!this.studentId) {
          this.$message.error('请输入学号!');
          return;
        }
        console.log(this.selectedModel);

        try {
          const base64Data = this.capturedImage.split(',')[1];
          const response = await this.$axios.post('/validate-visitor/', {
            image: base64Data,
            student_id: this.studentId,
            model: this.selectedModel
          });

          this.visitorInfo = response.data.visitor;
          this.message = response.data.result;
          this.isSuccess = true;
        } catch (error) {
          console.error('上传图片错误:', error);
          this.message = '签到失败';
          this.isSuccess = false;
        }
      },
      resetImage() {
        this.capturedImage = null;
        this.message = '';
        this.visitorInfo = null;
      }
    }
  };
  </script>
```

在查看签到记录模块中，使用 el-table 组件显示签到记录，表格包括学号、姓名和签到时间等信息。该模块提供了一个查看历史签到记录的界面。脚本部分定义了 visitorHistory 变量来存储签到记录数据，并在 mounted 生命周期钩子中发送 GET 请求到服务器以获取签到记录数据，获取到的数据存储到 visitorHistory 变量中进行展示。

```
<template>
  <div>
    <el-table :data="visitorHistory" style="width: 100%">
      <el-table-column prop="student_id" label="学号" width="180"></el-table-column>
      <el-table-column prop="visitor_name" label="姓名" width="180"></el-table-column>
      <el-table-column prop="timestamp" label="签到时间" width="180"></el-table-column>
    </el-table>
  </div>
</template>

<script>
export default {
  data() {
    return {
      visitorHistory: []
    }
  },
  async mounted() {
    try {
      const response = await this.$axios.get('history/')
      this.visitorHistory = response.data
    } catch (error) {
      console.error('Error fetching visitor history:', error)
    }
  }
}
</script>
```

主界面部分使用 el-container、el-header 和 el-main 组件创建主界面布局，使用 el-menu 组件创建导航菜单，包括人脸考勤、签到记录和添加用户三个菜单项。每个菜单项通过 router-link 组件实现页面跳转。脚本部分定义了 activeIndex 变量来存储当前激活的菜单项索引，并在 mounted 生命周期钩子中根据当前路由更新 activeIndex。通过$route 监听路由变化，动态更新 activeIndex，实现导航菜单的同步更新。

```
<template>
  <div id="app">
    <el-container>
      <el-header>
        <el-menu mode="horizontal" :default-active="activeIndex">
          <el-menu-item index="1">
            <router-link to="/">人脸考勤</router-link>
          </el-menu-item>
          <el-menu-item index="2">
            <router-link to="/history">签到记录</router-link>
          </el-menu-item>
          <el-menu-item index="3">
            <router-link to="/add">添加用户</router-link>
          </el-menu-item>
        </el-menu>
      </el-header>

      <el-main>
        <router-view />
      </el-main>
    </el-container>
  </div>
</template>

<script>
export default {
  data() {
    return {
      activeIndex: '1'
    }
  },
  watch: {
    '$route'(to) {
      this.updateActiveIndex(to)
    }
  },
  mounted() {
    this.updateActiveIndex(this.$route)
  },
  methods: {
    updateActiveIndex(route) {
      switch (route.path) {
        case '/':
          this.activeIndex = '1'
          break
        case '/history':
          this.activeIndex = '2'
          break
        case '/add':
          this.activeIndex = '3'
          break
        default:
          this.activeIndex = '1'
      }
    }
  }
}
</script>
```

3.4.2 后端代码编写

后端目录树如下：

```
backend
│   db.sqlite3
│   manage.py
│
├──backend
│   │   asgi.py
│   │   settings.py
│   │   urls.py
│   │   wsgi.py
│   │   __init__.py
│   │
│   └──__pycache__
│
├──media
│   └──visitor_images
│
├──model
│   │   urls.py
│   │
│   ├──model1
│   │
│   ├──model2
│   │   ├──anti_spoof_models
│   │   │
│   │   └──detection_model
│   │
│   └──model3
│
└──recognition
        admin.py
        apps.py
        models.py
        mtcnn3.py
        serializers.py
        tests.py
        test_model_2.py
        urls.py
        views.py
        __init__.py
        │
```

```
├──migrations
│   │
│   └──__pycache__
│
├──src
│   │   anti_spoof_predict.py
│   │   default_config.py
│   │   generate_patches.py
│   │   train_main.py
│   │   utility.py
│   │
│   ├──data_io
│   │   │   dataset_folder.py
│   │   │   dataset_loader.py
│   │   │   functional.py
│   │   │   transform.py
│   │   │
│   │   └──__pycache__
│   │
│   ├──model_lib
│   │   │   MiniFASNet.py
│   │   │   MultiFTNet.py
│   │   │
│   │   └──__pycache__
│   │
│   └──__pycache__
│
└──__pycache__
```

后端相应的也分为三个模块：添加用户、签到记录以及人脸实时考勤。

首先，设置好数据库结构。每个用户有唯一的学号及对应的名字、人脸图片及人脸特征，签到记录则记录着用户的姓名及其签到时间。

```python
class Visitor(models.Model):
    student_id = models.CharField(max_length=20, unique=True)
    name = models.CharField(max_length=255)
    face_image = models.ImageField(upload_to='visitor_images/', unique=True)
    face_features = models.BinaryField(null=True, blank=True)

    def __str__(self):
        return f"{self.name} ({self.student_id})"


5个用法
class VisitHistory(models.Model):
    visitor = models.ForeignKey(Visitor, on_delete=models.CASCADE)
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Visit by {self.visitor.name} on {self.timestamp}"
```

在添加用户模块中，定义了一个 VisitorView 类，继承自 APIView。该类的 post 方法用于处理添加访客的请求。首先，通过 VisitorSerializer 验证请求数据的有效性。如果验证通过，保存访客信息并读取上传的脸部图像。使用 OpenCV 将图像转换为 RGB 格式，并通过人脸检测器 face_detector 检测图像中的人脸。如果检测到人脸，提取人脸特征并将其保存到访客记录中。如果未检测到人脸，返回错误信息。

```
# 使用numpy读取中文路径的图像
1个用法
def cv_imread(file_path):
    cv_img = cv2.imdecode(np.fromfile(file_path, dtype=np.uint8), cv2.IMREAD_COLOR)
    return cv_img


# 添加用户模块
2个用法
class VisitorView(APIView):
    def post(self, request):
        serializer = VisitorSerializer(data=request.data)
        if serializer.is_valid():
            visitor = serializer.save()
            face_image_path = visitor.face_image.path
            image = cv_imread(face_image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            face_detector.setInputSize((image.shape[1], image.shape[0]))
            _, faces = face_detector.detect(image)

            if faces is None or len(faces) == 0:
                return Response( data: {'error': '未检测到人脸。'}, status=status.HTTP_400_BAD_REQUEST)

            # 人脸特征提取
            face_aligned = face_recognizer.alignCrop(image, faces[0])
            face_feature = face_recognizer.feature(face_aligned)

            # 展平并保存特征
            face_feature_flat = face_feature.flatten() if len(face_feature.shape) > 1 else face_feature
            visitor.face_features = face_feature_flat.tobytes()
            visitor.save()

            return Response(serializer.data, status=status.HTTP_201_CREATED)
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

在获取签到记录模块中，定义了一个 VisitHistoryView 类，继承自 generics.ListAPIView。该类用于获取所有访客的签到记录，并按照时间降序排列。使用 VisitHistorySerializer 对签到记录进行序列化。

```
# 签到记录
2个用法
class VisitHistoryView(generics.ListAPIView):
    queryset = VisitHistory.objects.all().order_by('-timestamp')  # 按时间降序排列
    serializer_class = VisitHistorySerializer
```

在验证访客模块中，定义了一个 ValidateVisitorAPIView 类，继承自 APIView。该类的 post 方法用于处理访客验证请求。首先，从请求中获取图像数据和学号，并根据所选模型进行检测。支持三种模型：MobileNetV2、MTCNN 结合 MiniFASNet、以及第三个模型。每个模型都有相应的预测方法。

```
class ValidateVisitorAPIView(APIView):
    def post(self, request):
        try:
            image_data = request.data.get('image', '')
            student_id = request.data.get('student_id', '')
            model_choice = request.data.get('model', 'model1')
```

```
        if not image_data or not student_id:
            return JsonResponse( data: {'error': '缺少图像或学生ID。'}, status=400)

        # 解码并转换图像
        image = base64.b64decode(image_data)
        image = Image.open(io.BytesIO(image))
        image_np = np.array(image)
        image_np = cv2.cvtColor(image_np, cv2.COLOR_RGB2BGR)

        # 根据所选模型执行检测
        if model_choice == 'model1':
            resized_image = cv2.resize(image_np, (224, 224))
            img_array = resized_image.astype('float32') / 255.0
            img_array = np.expand_dims(img_array, axis=0)
            prediction = liveness_model.predict(img_array)[0][0]

            print(prediction)
            if prediction <= 0.4:
                return JsonResponse( data: {'result': '拒绝，不是真人。'}, status=403)

        elif model_choice == 'model2':
            prediction = model2_predict(image_np)
            print(prediction)
            if prediction == 1:
                return JsonResponse( data: {'result': '拒绝，不是真人。'}, status=403)

        elif model_choice == 'model3':
            result = model3_predict(image_np)
            if result == "Fake" or result == "No Face Detected":
                return JsonResponse( data: {'result': f'拒绝，{result}。'}, status=403)

        else:
            return JsonResponse( data: {'error': '无效的模型选择。'}, status=400)
```

对于第一个模型，即 MobileNetV2 模型，将图像调整为 224x224 像素，并进行归一化处理，使用预训练的 MobileNetV2 模型进行预测。如果预测结果显示为假脸，返回拒绝信息。

```
# 初始化模型1
1 个用法
def load_mobilenet_model():
    model_path = './model/model1/mobilenetv2-best.hdf5'
    base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights=None)
    x = base_model.output
    x = Conv2D( filters: 32,  kernel_size: (3, 3), activation='relu')(x)
    x = Dropout(0.2)(x)
    x = GlobalAveragePooling2D()(x)
    output = Dense( units: 1, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=output)
    model.load_weights(model_path)
    return model

liveness_model = load_mobilenet_model()

# OpenCV 面部检测和识别
face_detector = cv2.FaceDetectorYN.create('./model/model1/face_detection.onnx', '', (320, 320), 0.9, 0.3, 5000)
face_recognizer = cv2.FaceRecognizerSF.create('./model/model1/face_recognition.onnx', '')
```

对于第二个模型，即 MTCNN 结合 MiniFASNet 模型，使用 MTCNN 检测人脸，并将人脸区域传递给活体检测模型 test 进行预测。如果预测结果显示为假脸，返回拒绝信息。

```python
# 初始化模型2
# 使用 MTCNN 和 InceptionResnetV1 初始化
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
mtcnn = MTCNN(keep_all=True, device=device)
resnet = InceptionResnetV1(pretrained='vggface2').eval().to(device)

model_dir = "./model/model2/anti_spoof_models"

1个用法
def model2_predict(image_np):
    """利用第二个模型进行活体检测"""
    frame_rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
    boxes, probs = mtcnn.detect(frame_rgb)

    if boxes is not None and len(boxes) > 0:
        for box, prob in zip(boxes, probs):
            print(prob)
            if prob > 0.8:
                face = frame_rgb[int(box[1]):int(box[3]), int(box[0]):int(box[2])]
                face = cv2.resize(face, (128, 128))

                spoof_label = test(face, model_dir, device_id: 0)
                print()
                return 0 if spoof_label == 0 else 1
    return 1
```

对于第三个模型，将整张图像调整为模型输入大小，并进行归一化处理，使用预训练的模型进行预测。如果预测结果显示为假脸，返回拒绝信息。

```python
# 初始化模型3
model3_path = "./model/model3/fas.h5"
model3 = load_model(model3_path)
mtcnn_model3 = MTCNN3(
    model_path: './model/model3/mtcnn.pb',
    min_size=40,  # 适当降低最小尺寸
    factor=0.7,   # 调整缩放因子
    thresholds=[0.6, 0.7, 0.7]  # 调整检测阈值
)


1个用法
def model3_predict(image_np):
    """利用模型3进行检测，将整张图像调整为模型输入大小"""
    image_resized = cv2.resize(image_np, (224, 224))
    print(f"Image resized shape: {image_resized.shape}")

    image_standardized = (image_resized - 127.5) / 127.5
    input_data = np.expand_dims(image_standardized, axis=0)

    score = model3.predict(input_data)[0]
    print(f"Score: {score}")

    return "Real" if score > 0.1 else "Fake"
```

在通过模型验证后，使用人脸检测器和识别器对图像进行处理。首先，检测图像中的人脸，并提取人脸特征。然后，从数据库中获取对应学号的访客信息，并将提取的特征与数据库中的特征进行比较。如果相似度低于阈值，返回不匹配信息。如果匹配成功，记录此次访问并返回成功信息。

```python
            # 通用人脸检测与验证逻辑
            face_detector.setInputSize((image_np.shape[1], image_np.shape[0]))
            _, faces = face_detector.detect(image_np)

            if faces is None or len(faces) == 0:
                return JsonResponse( data: {'result': '未检测到人脸。'}, status=404)
```

```
            face_aligned = face_recognizer.alignCrop(image_np, faces[0])
            face_feature = face_recognizer.feature(face_aligned)

            try:
                visitor = Visitor.objects.get(student_id=student_id)
                stored_face_feature = np.frombuffer(visitor.face_features, dtype=np.float32)
                face_feature_flat = face_feature.flatten() if len(face_feature.shape) > 1 else face_feature
                stored_face_feature = stored_face_feature.reshape(face_feature_flat.shape)

                similarity = np.dot(stored_face_feature, face_feature_flat) / (
                    np.linalg.norm(stored_face_feature) * np.linalg.norm(face_feature_flat))

                print(similarity)
                if similarity < 0.45:
                    return JsonResponse( data: {'result': '面部与数据库不匹配。'}, status=403)

                VisitHistory.objects.create(visitor=visitor)
                return JsonResponse({'result': '成功认证。', 'visitor': VisitorSerializer(visitor).data})

            except Visitor.DoesNotExist:
                return JsonResponse( data: {'result': '数据库中未找到访客。'}, status=404)

        except Exception as e:
            error_trace = traceback.format_exc()
            print(f"ValidateVisitorAPIView 错误: {e}\n{error_trace}")
            return JsonResponse( data: {'error': f'错误: {str(e)}'}, status=500)
```

3.5 效果展示

前端运行 npm run dev，网站部署在本地网络的 5173 端口。

```
PS D:\Content_Secu\task6\frontend> npm run dev

> front_end@0.0.0 dev
> vite

Re-optimizing dependencies because vite config has changed

  VITE v5.2.11  ready in 3367 ms

  ➜  Local:   http://127.0.0.1:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

后端运行 python manage.py runserver，部署在本地网络的 8000 端口。

```
System check identified no issues (0 silenced).
May 21, 2024 - 10:08:29
Django version 4.2.11, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

添加用户页面

用户输入新的学号及姓名，并上传相应的人脸图片，即可添加用户到数据库。

人脸考勤页面

网页会请求摄像头权限，用户截取摄像头画面，并确认上传该画面，具体效果见三个视频文件。

签到记录页面

用户签到成功后，会有相应的签到记录，记录按时间逆序显示。

四、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）

问题：前后端交互存在问题。
解决方法：前端使用 axios 进行跨域访问，后端使用 CORS 允许跨域。

五、个人小结

（描述实验心得，可提出实验的改进意见）

在这次实验中，我设计并实现了一个基于人脸识别和活体检测技术的班级考勤系统。整个系统采用 BS 架构，服务端使用 Django 框架，前端使用 Vue.js 框架。实验过程中，我深入了解了多个深度学习模型的应用，并结合实际需求进行调整和优化，最终实现了一个能够有效抵抗照片和视频攻击的考勤系统。

首先，我对比和研究了几种常用的人脸识别和活体检测技术。最终选择了 MobileNetV2、MTCNN 结合 MiniFASNet 以及一个自定义的活体检测模型。MobileNetV2 具有较高的计算效率和较低的计算成本，适合实时处理；MTCNN 结合 InceptionResnetV1 在准确度上表现优异，尤其在检测部分遮挡和光照变化的人脸时表现突出；自定义的活体检测模型则通过多尺度特征融合和傅里叶变换辅助监督，进一步提升了对假脸攻击的检测能力。

为了确保系统的稳定性和可扩展性，服务端采用 Django 框架，前端使用 Vue.js，依赖库包括 OpenCV、TensorFlow、Keras、facenet_pytorch 等。此外，我还在实验过程中使用了虚拟环境来管理和隔离项目的依赖包，确保各个依赖库版本的兼容性。

我首先完成了添加用户的功能。使用前端表单收集用户的姓名、学号和上传的面部照片，并将这些信息存储在数据库中。为了提高系统的安全性和准确性，我在用户注册时加入了人脸特征提取功能。使用人脸检测器检测上传的面部照片，提取人脸特征并存储到数据库中。

接下来，我实现了人脸识别考勤功能。用户通过前端界面启动摄像头，实时捕捉面部图像，并将图像传输到服务端进行处理。服务端接收到图像后，首先进行活体检测，判断图像是否为真实人脸。对于通过活体检测的图像，进一步进行人脸识别，将提取的人脸特征与数据库中的特征进行比对，判断用户身份并记录考勤信息。

为了确保系统的鲁棒性，我引入了多种活体检测方案。用户可以在前端界面选择不同的检测模型，包括基于 MobileNetV2 的模型、MTCNN 结合 MiniFASNet 的模型以及自定义的活体检测模型。不同的模型在不同的场景下具有各自的优势，通过多模型的融合使用，进一步提升了系统的安全性和准确性。

在实验结果分析阶段，不同的光照条件对于人脸识别考勤的准确性有一定影响。同时，在防止照片和视频攻击方面，系统表现出了较强的鲁棒性。对于不同的假脸攻击方式，如打印的照片、屏幕翻拍、3D 模型等，系统均能够有效检测并拒绝考勤。

在实现人脸特征比对功能时，由于不同设备拍摄的图像分辨率和质量差异较大，导致特征提取和比对的准确性受到影响。为了解决这一问题，我对图像预处理流程进行了调整，引入了图像标准化和对齐方法，确保特征提取的一致性。

这次实验中，我不仅掌握了人脸识别和活体检测的基本原理和实现方法，还深刻理解体会了深度学习模型在实际应用中的调整和优化过程。此外，我也认识到在实际项目中，系统的鲁棒性和安全性是非常重要的，只有通过不断的优化和调整，才能实现一个稳定、高效的系统。

六、教师评语及评分

教师签名：

年　　月　　日