第8章 计算机视觉处理

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 人工智能基础与应用 | | 章名 | | | 计算机视觉处理 | |
| 教学内容 | 计算机视觉处理2 | | | | 课时 | 2 | |
| 项目性质 | □演示性 □验证性 □设计性 √综合性 | | | | | | |
| 授课班级 |  | 授课日期 | |  | | 授课地点 |  |
| 教学目标 | 掌握人脸识别项目的实现 | | | | | | |
| 教学内容 | 人脸识别 | | | | | | |
| 教学重点 | 基于face\_recognition项目实现人脸识别 | | | | | | |
| 教学难点 | 基于face\_recognition项目实现人脸识别 | | | | | | |
| 教学准备 | 装有Python的计算机  教学课件PPT  教材：《人工智能基础与应用（微课版）》 | | | | | | |
| 作业设计 |  | | | | | | |

教学过程

|  |  |
| --- | --- |
| **教学环节** | **教学内容与过程**  **（教学内容、教学方法、组织形式、教学手段）** |
| **课前组织** | 做好上课前的各项准备工作（打开计算机、打开课件、打开软件、打开授课计划、教案等），吸引学生注意力。 |
| **课程说明** | 【课前说明】  回顾计算机视觉开发和图像识别相关知识点。  【目的】  使学生从了解本节课的学习目标、学习重点、考评方式等方面明确课程学习的要求和目标。 |
| **课程内容描述** | 8.3 人脸识别  人脸识别是基于人的脸部特征信息进行身份识别的一种生物识别技术，如今很多场景下都使用人脸识别进行安全验证。由于人脸识别具有并发性、非接触性以及非强制性，所以运用十分广泛，如智能手机的人脸识别开锁、火车站的刷脸进站、支付宝的刷脸支付、北京大兴国际机场的刷脸登机等场景都应用了人脸识别技术。智能手机的刷脸解锁速度已经可以做到从口袋拿出手机后看一眼手机就完成上划解锁，甚至让人感觉不到有人脸验证解锁这一步。支付宝的刷脸支付货架如图所示。    支付宝的刷脸支付货架  8.3.1 项目介绍  传统人脸识别需要人工提取特征，但是由于受到光照、角度等的影响会造成识别率不高、识别速度不快等问题，深度学习现在成为了人脸识别的主流方法，无须人工提取特征。人脸识别系统包含目标检测与图像分类的过程，目标检测就是在图像中找到人脸，图像分类就是识别人脸。  本项目使用读取图像以及调用摄像头两种方式完成图像中人脸检测、人脸关键点检测、人脸对比、人脸搜索与人脸识别。  本项目基于face\_recognition项目开发。face\_recognition项目的人脸识别是基于C++开源库dlib中的深度学习模型实现的，用LFW（Labeled Faces in the Wild Home）人脸数据集进行测试时，准确率可达到99.38%。  在项目开始之前需要安装face\_recognition第三方库：Python版本需要在Python 3.3及以上或者Python 2.7上安装；在Mac、Linux或者Windows上安装时，首先需要安装dlib，然后在交互界面输入“pip3 install face\_recognition”命令安装项目源码，或者直接在GitHub网站下载项目源码。  8.3.2 人脸的数据集介绍  人脸数据集很多，本书介绍LFW（Labeled Faces in the Wild Home）数据集。该数据集是由美国马萨诸塞州立大学阿默斯特分校计算机视觉实验室整理完成的，主要用来研究非受限情况下的人脸识别问题。在LFW数据集中，由于多姿态、光照、表情、年龄、遮挡等因素的影响，即使是同一人的照片差别也很大。并且有些照片中可能存在多张人脸，对这些存在多张人脸的图像，仅选择中心坐标的人脸作为识别目标，其他区域的内容视为背景干扰。该数据集包含5746个人的13233张照片，其中1680个人拥有多张照片，每张照片的大小为250像素×250像素。  下载LFW数据集，解压后的目录如图所示。    LFW数据集解压后的目录  LFW数据集文件夹按名字命名，人脸照片在每个名字的文件夹下，人脸照片命名方式为“名字\_xx.jpg”，如Aaron\_Eckhart\_0001.jpg。  8.3.3 人脸识别流程  人脸识别流程主要包括4个组成部分，分别为人脸图像采集及检测、人脸图像预处理、人脸图像特征提取以及匹配与识别。  1．人脸图像采集及检测  人脸识别首先需要采集人脸图像，可以通过读取图像或者通过摄像头直接采集来完成。某幅图像上可能存在人脸，也可能不存在人脸，所以需要进行人脸检测。人脸检测在实际应用中主要用于人脸识别的预处理，即在图像中准确标出人脸的位置和大小。人脸图像中包含的模式特征十分丰富，如直方图特征、颜色特征、模板特征、结构特征及Haar特征等。人脸检测就是把其中有用的信息挑出来。  2．人脸图像预处理  在人脸检测之后，将人脸图像截取出来，然后对该图像进行预处理，包括人脸图像的光线补偿、灰度变换、直方图均衡化、归一化、几何校正、滤波以及锐化等。预处理可以尽量避免环境条件限制和随机干扰，提高特征提取准确率。  3．人脸图像特征提取  人脸图像特征提取也称为人脸表征，它是对人脸进行特征建模的过程，可以将图像信息数字化，根据人脸器官的形状描述以及它们之间的距离特性来获得有助于人脸分类的特征数据。其特征分量通常包括特征点间的欧氏距离、曲率和角度等。人脸由眼睛、鼻子、嘴、下巴等局部部位构成，这些局部部位之间结构关系的几何描述可作为识别人脸的重要特征。  4．匹配与识别  匹配是将所提取的人脸图像的特征数据与数据库中存储的特征模板进行搜索匹配，通常设定一个阈值，当相似度超过这一阈值时，则把匹配得到的结果输出。人脸识别就是将待识别的人脸特征与已得到的人脸特征模板进行比较，根据相似程度对人脸的身份信息进行判断。这一过程又分为两类：一类是确认，是一对一进行图像比较的过程，如手机解锁；另一类是辨认，是一对多进行图像匹配对比的过程，如公共安全监测。  8.3.4 人脸识别方案  人脸识别是基于人的脸部特征信息进行身份识别的一种生物识别技术，是用摄像机或摄像头采集含有人脸的图像或视频流，并自动在图像中检测和跟踪人脸，进而对检测到的人脸进行脸部识别的一系列相关技术，通常也称为人像识别、面部识别。  1．怎么找到人脸  人脸识别的第一步是找到人脸，即人脸检测，在一张照片或一个视频帧中，首先要知道是否存在人脸以及人脸的位置。在智能手机或者相机上拍照时，人脸检测可以确保在拍摄时对焦到所有人脸。在人脸识别系统中，它的作用是将人脸区域传递到预处理中。  找到人脸之后需要提取整体图像特征，提取特征的方法有方向梯度直方图（Histogram of Oriented Gradient，HOG）、局部二值（Local Binary Pattern，LBP）以及Haar-like。  OpenCV中对AdaBoost与Haar-like组成的级联人脸检测做了封装。在深度学习被广泛运用之后，人脸检测更多地使用CNN或者其他网络完成。face\_recognition可以使用方向梯度直方图或者更为精确的基于深度学习的面部检测模型检测照片中的人脸，但是使用深度学习的面部检测模型需要GPU加速（通过NVIDIA的CUDA库）才能获得良好的性能。  2．简单的面部识别分类  根据之前的步骤将脸部从图像中分离，如果直接将两张照片进行对比，当两者中人脸的角度、位置不同时，接下来的网络或者算法在做分类时准确率降低，所以通常需要先对脸部图像进行预处理。比较通用的预处理方法是瓦希德·卡泽米（Vahid Kazemi）和约瑟芬·沙利文（Josephine Sullivan）提出的面部特征点估计，该方法的主要思路是找到面部中普遍存在的68个特征点，包括下巴、每只眼睛的外部轮廓、每条眉毛的内部轮廓等，然后基于这些特征点的位置对图像进行仿射变换等操作，让人脸尽量居中。  脸部居中之后可以进行识别，最简单的方法是将要识别的人脸与数据库被标注的人脸进行比较，看是否相似。直接比较的话，如果人脸数据库中有上万张甚至十几万张照片，那么逐个比较会需要很长的时间。embedding可以解决这个问题，这种方法是将图像这种复杂的数据生成一个简单的数列，具体思路是：使用卷积神经网络进行面部编码，将面部图像在网络中进行训练并生成128个参数值。例如，现在有3张照片，照片A和照片C属于同一个人，照片B属于另外一个人，设计神经网络，使3张照片学习并生成3组参数值，其中，照片A和照片C生成的参数值相近，且与照片B生成的参数值有差距，经过多次学习后，即使某个人有很多照片，神经网络也可以为这个人生成一个大概范围的参数值。  这个卷积神经网络的作用是将某个人的人脸照片进行编码。使用这个网络，可以对新的需要人脸识别的用户进行编码。  最后一步就是人脸识别，有了前面的铺垫，这一步就很简单了。得到需要识别的人脸并将其编码之后，使用分类算法就可以完成识别，如KNN。需要注意的是，这里的KNN并不是对比两张照片的像素距离，而是对比编码后的128个参数值的距离。  8.3.5 人脸识别应用  基于人脸技术可以完成多种应用，包括人脸检测、人脸关键点检测、人脸对比、人脸搜索、人脸识别等。  8.3.4小节介绍了一种人脸识别的原理，本小节基于该原理的face\_recognition项目实现人脸检测、人脸关键点检测、人脸对比、人脸搜索以及人脸识别。  1．人脸检测  在CV目录下新建face\_predict项目目录，同时在face\_predict目录下新建Face\_database作为人脸库目录，在Face\_database目录下新建以人名命名的人脸库子目录，可以新建多个人脸库子目录，每个人脸库子目录下可以存放某人的多张照片。  【例8-5】 在face\_predict下新建face-find.py文件，读取照片并将检测后的人脸标注出来。  import cv2  import face\_recognition  # 加载被比较的图像  frame = face\_recognition.load\_image\_file("Face\_database/hyz/hyz.png")  # 使用CPU获得人脸边界框的数列  face\_locations = face\_recognition.face\_locations(frame)  # 使用CNN并利用GPU/CUDA加速获得人脸边界框的数列  # 相对更准确  # face\_locations = face\_recognition.face\_locations(frame, number\_of\_times\_to\_ upsample=0, model="cnn")  print("该张图像中有 {} 张人脸。".format(len(face\_locations)))  # 圈出人脸边界框  for (top, right, bottom, left) in face\_locations:  cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)  # 显示得到人脸后的图像  frame = frame[:, :, ::-1]  cv2.imshow("image", frame)  cv2.waitKey(0)  本段代码使用非CNN的方式检测人脸，读取Face\_database目录下hyz人脸库子目录下的图像，读取“hyz.png”图像，使用face\_recognition.face\_locations()函数寻找人脸，将照片中的人脸数量输出，并将人脸用矩形框圈出，人脸检测图像结果如图所示。    人脸检测图像结果  2．人脸关键点检测  人脸关键点检测是给定人脸图像，定位出人脸面部的关键区域，包括眉毛、眼睛、鼻子、嘴巴、脸部轮廓等。  【例8-6】 在face\_predict下新建face-feature.py文件，读取照片并标记特征点。  # 加载被比较的图像  frame = face\_recognition.load\_image\_file("Face\_database/hyz/hyz.png")  # 查找图像中的所有面部特征  face\_landmarks\_list = face\_recognition.face\_landmarks(frame, face\_locations = None, model ='large')  # 查找图像中的鼻子、左眼、右眼面部特征  # face\_landmarks\_list = face\_recognition.face\_landmarks(frame, face\_locations= None, model='small')  print("该张图像中有 {} 张人脸。".format(len(face\_landmarks\_list)))  for face\_landmarks in face\_landmarks\_list:  # 打印此图像中每个面部特征的位置  # 查找图像中所有面部特征的列表  facial\_features = [  'chin',  'left\_eyebrow',  'right\_eyebrow',  'nose\_bridge',  'nose\_tip',  'left\_eye',  'right\_eye',  'top\_lip',  'bottom\_lip'  ]  # 查找图像中鼻子、左眼、右眼面部特征的列表  # facial\_features = [  # 'nose\_tip',  # 'left\_eye',  # 'right\_eye',  # ]  # 在图像中描绘出人脸特征  for facial\_feature in facial\_features:  # 数据类型必须是int32  pts = np.array(face\_landmarks[facial\_feature], np.int32)  pts = pts.reshape((-1, 1, 2))  # 图像，点集，是否闭合，颜色，线条粗细  cv2.polylines(frame, [pts], False, (0, 0, 0), 2)  # 显示得到人脸后的图像  frame = frame[:, :, ::-1]  cv2.imshow("image", frame)  cv2.waitKey(0)  读取图像，查找图像中的面部特征，并且将所有的特征描绘出来，包括下巴、左眼眉毛、右眼眉毛、鼻梁、左眼、右眼、上嘴唇以及下嘴唇，人脸关键点检测结果如图所示。    人脸关键点检测结果  3．人脸对比  人脸对比就是计算两张脸的相似程度，并给出相似度评分，以便分析两张脸属于一个人的可能性。比较相似度实际上就是比较两张人脸编码之后参数值的距离。人脸对比常用于需要进行人脸验证的场合，过程是输入两张人脸，编码后进行运算，输入比对阈值，根据是否超过这个阈值判断两张人脸是否属于同一个人。  【例8-7】 在face\_predict下新建face-compare.py，完成人脸对比。  # 人脸比较：将两张人脸图像进行对比  # 将两者之间的相似值进行打印  # 阈值为0.6，阈值越小，条件越苛刻  import cv2  import face\_recognition  # 加载被比较的图像  source\_image = face\_recognition.load\_image\_file("Face\_database/hyz/hyz.png")  # 加载测试图像  compare\_image = face\_recognition.load\_image\_file("Face\_database/hyz/hyz\_ near.png")  # 获取人脸位置并做单人脸容错处理  source\_locations = face\_recognition.face\_locations(source\_image)  if len(source\_locations) != 1:  print("注意：图像一只能有一张人脸哦！")  exit(0)  # 获取人脸位置并做单人脸容错处理  compare\_locations = face\_recognition.face\_locations(compare\_image)  if len(compare\_locations) != 1:  print("注意：图像二只能有一张人脸哦！")  exit(0)  # 绘制图像一的人脸  for (top, right, bottom, left) in source\_locations:  print(top, right, bottom, left)  cv2.rectangle(source\_image, (left, top), (right, bottom), (0, 255, 0), 2)  # 绘制图像二的人脸  for (top, right, bottom, left) in compare\_locations:  print(top, right, bottom, left)  cv2.rectangle(compare\_image, (left, top), (right, bottom), (0, 255, 0), 2)  # 获取图像一的面部编码  source\_face\_encoding = face\_recognition.face\_encodings(source\_image)[0]  source\_encodings = [  source\_face\_encoding,  ]  # 获取图像二的面部编码  compare\_face\_encoding = face\_recognition.face\_encodings(compare\_image)[0]  # 显示两张得到人脸后的图像  source\_image = source\_image[:, :, ::-1]  cv2.imshow("image", source\_image)  cv2.waitKey(0)  compare\_image = compare\_image[:, :, ::-1]  cv2.imshow("image", compare\_image)  cv2.waitKey(0)  # 查看面部一与面部二的比较结果，阈值为0.6，阈值越小越苛刻  face\_distances = face\_recognition.compare\_faces(source\_encodings, compare\_face\_ encoding, 0.6)  # 输出结果  print("正常阈值为0.6时，测试图像是否与已知图像{}匹配!".format("是" if face\_distances else "不是"))  将两张照片图像加载进来后，先判断照片中是否只有一张人脸。如果照片中多于一张人脸，提示并退出；如果两张照片中都只有一张人脸，那么就将人脸圈出来，圈出来的人脸进行编码后进行比对，阈值为0.6，小于该阈值，就认为两张图像属于一张人脸。  4．人脸搜索  人脸搜索是基于人脸对比的，将需要加入人脸库的照片放入Face\_database的子目录下，与人脸对比的过程类似，将unknown目录下的未知图像与人脸库中的照片编码后逐一进行对比，如果结果全部小于阈值，就认为在人脸库中查找到该未知人脸的归属。  【例8-8】 在face\_predict下新建face-seek.py，完成人脸搜索。  # 查找人脸：查找图像中的人脸并标记出来  import os  import face\_recognition  file\_name = []  known\_faces = []  # 加载文件中的人脸库图像  image\_dir = "Face\_database/hyz/"  for parent, dirnames, filenames in os.walk(image\_dir):  for filename in filenames:  # print(filename)  # 加载图像  frame = face\_recognition.load\_image\_file(image\_dir + filename)  face\_bounding\_boxes = face\_recognition.face\_locations(frame)  if len(face\_bounding\_boxes) != 1:  # 如果训练图像中没有人（或人太多），请跳过图像  print("{} 这张图像不适合训练: {}。".format(image\_dir + filename, "因为它上面没找到人脸" if len(face\_bounding\_boxes) < 1 else "因为它不止一张人脸"))  else:  # encoding  frame\_face\_encoding = face\_recognition.face\_encodings(frame)[0]  # 加到列表里  known\_faces.append(frame\_face\_encoding)  file\_name.append(filename)  # 加载未知图像  frame = face\_recognition.load\_image\_file("unknown/unknown1.png")  # encoding  frame\_face\_encoding = face\_recognition.face\_encodings(frame)[0]  # 比较获得结果  results = face\_recognition.compare\_faces(known\_faces, frame\_face\_encoding)  print(results)  首先读取Face\_database子目录下的人脸库图像并进行编码，当图像中没有人或者有超过一个人时，就跳过该图像，将编码后的参数以及名字放入列表中，然后读取未知图像，编码后进行比对，如果比对结果返回True，则表示匹配成功，代表该未知图像中的人存在于人脸库中，并输出人脸库中的名字。接下来比对下一个人脸库中的图像，如果全部比对后没有匹配成功，则说明人脸库中没有该未知图像中的人。  5．人脸识别  人脸识别中，人脸编码过程与人脸对比完全一致。识别过程是用KNN完成的，KNN分类器首先对人脸库进行训练，训练完成后得到训练模型，在预测时调用该模型，然后与未知图像一起使用KNN完成预测分类。需要使用scikit-learn第三方库的KNN库进行模型训练，训练过程其实就是将图像编码并存储的过程，可以使用pip安装scikit-learn。  【例8-9】 在face\_predict下新建face-knn-train.py，使用KNN实现人脸库的训练。  # 训练K近邻分类器  import math  from sklearn import neighbors  import os  import os.path  import pickle  import face\_recognition  from face\_recognition.face\_recognition\_cli import image\_files\_in\_folder  def train(train\_dir, model\_save\_path=None, n\_neighbors=None, knn\_algo='ball\_ tree', verbose=False):  """  训练K近邻分类器进行人脸识别  param train\_dir：包含每个已知人员的子目录及人员名称的目录  param model\_save\_path：（可选）将模型保存在磁盘上的路径  param n\_neighbors：（可选）在分类中称重的邻居数。如果未指定，则自动选择  param knn\_algo：（可选）支持knn.default的底层数据结构是ball\_tree  param verbose：训练时是否根据图像数量取n\_neighbors的值  return：返回在给定数据上训练的KNN分类器  """  X = []  y = []  # 循环遍历训练集中的每个人  for class\_dir in os.listdir(train\_dir):  # 如果train\_dir/class\_dir不是一个目录，就继续  if not os.path.isdir(os.path.join(train\_dir, class\_dir)):  continue  # 循环浏览当前人员的每个训练图像  for img\_path in image\_files\_in\_folder(os.path.join(train\_dir, class\_dir)):  image = face\_recognition.load\_image\_file(img\_path)  face\_bounding\_boxes = face\_recognition.face\_locations(image)  if len(face\_bounding\_boxes) != 1:  # 如果训练图像中没有人（或人太多），请跳过图像  if verbose:  print("{} 这张图像不适合训练: {}。".format(img\_path, "因为它上面没找到人脸" if len(face\_bounding\_boxes) < 1 else "因为它不止一张人脸"))  else:  # 将当前图像的面部编码添加到训练集  X.append(face\_recognition.face\_encodings(image, known\_face\_locations=face\_bounding\_boxes)[0])  y.append(class\_dir)  # 确定KNN分类器中用于加权的近邻  if n\_neighbors is None:  n\_neighbors = int(round(math.sqrt(len(X)))) # 面部编码长度开平方后四舍五入取整数  if verbose:  print("自动选择n\_neighbors:", n\_neighbors)  # 创建并训练KNN分类器  knn\_clf = neighbors.KNeighborsClassifier(n\_neighbors=n\_neighbors, \  algorithm=knn\_algo,  weights='distance')  knn\_clf.fit(X, y)  # 保存训练后的KNN分类器  if model\_save\_path is not None:  with open(model\_save\_path, 'wb') as f:  pickle.dump(knn\_clf, f)  return knn\_clf  if \_\_name\_\_ == "\_\_main\_\_":  # 训练的KNN分类，并将其保存到磁盘  print("训练KNN分类器...")  classifier = train("Face\_database", model\_save\_path="trained\_knn\_model.clf",\  n\_neighbors=1)  print("训练完成！")  训练完成后，会在face\_predict目录下生成trained\_knn\_model.clf模型。有了训练模型，接下来就可以对未知图像进行预测，预测时使用OpenCV调用摄像头，并且做实时的人脸识别。在face\_predict下新建face-knn-predict.py文件，实现人脸识别。  # 摄像头测试K近邻分类器  import os  import cv2  import os.path  import pickle  import face\_recognition  ALLOWED\_EXTENSIONS = {'png', 'jpg', 'jpeg'}  def predict(X\_img, knn\_clf=None, model\_path=None, distance\_threshold=0.6):  """  使用训练后的KNN分类器识别给定图像中的面部  param X\_img：要识别的图像  param knn\_clf：（可选）一个KNN分类器对象。如果未指定，则必须指定model\_save\_path  param model\_path：（可选）pickle KNN分类器的路径。如果未指定，则model\_save\_path必须为knn\_clf  param distance\_threshold：（可选）面部分类的距离阈值。它越大，机会就越大，就会将一个不知名的人误分类为已知人员  图像中已识别面部的名称和面部位置列表：[（名称，边界框），...]。对于未被识别人员的面孔，将返回“未知”的名称  """  if knn\_clf is None and model\_path is None:  raise Exception("必须提供KNN分类器knn\_clf或model\_path")  # 加载训练后的KNN模型（如果传入了一个）  if knn\_clf is None:  with open(model\_path, 'rb') as f:  knn\_clf = pickle.load(f)  # 加载图像并查找面部位置  X\_face\_locations = face\_recognition.face\_locations(X\_img)  # X\_face\_locations = face\_recognition.face\_locations(X\_img, number\_of\_ times\_to\_upsample=0, model="cnn")  # 如果图像中未找到面，则返回空结果  if len(X\_face\_locations) == 0:  print("没有检测到人脸！")  return []  # 在测试image中查找面部的编码  faces\_encodings = face\_recognition.face\_encodings(X\_img, known\_face\_ locations=X\_face\_locations)  # 使用KNN模型找到测试的最佳匹配  # 找到一个点的K近邻，返回每个点的邻居的索引和距离  closest\_distances = knn\_clf.kneighbors(faces\_encodings, n\_neighbors=1)  # print(closest\_distances)  are\_matches = [closest\_distances[0][i][0] <= distance\_threshold for i in  range(len(X\_face\_locations))]  # print(are\_matches)  # 预测类并删除不在阈值范围内的分类  # predict：返回分类的标签  return [(pred, loc) if rec else ("unknown", loc) for pred, loc, rec in  zip(knn\_clf.predict(faces\_encodings), X\_face\_locations, are\_matches)]  if \_\_name\_\_ == "\_\_main\_\_":  video\_capture = cv2.VideoCapture(0)  while True:  ret, frame = video\_capture.read()  small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)  predictions = predict(small\_frame, model\_path="trained\_knn\_model.clf")  for name, (top, right, bottom, left) in predictions:  top \*= 4  right \*= 4  bottom \*= 4  left \*= 4  cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)  cv2.putText(frame, name, (left + 6, bottom - 6), cv2.FONT\_HERSHEY\_ DUPLEX, 1.0, (255, 255, 255), 1)  cv2.imshow('Video', frame)  if cv2.waitKey(1) & 0xFF == ord('q'):  break  video\_capture.release()  cv2.destroyAllWindows()  调用摄像头实时检测人脸并进行识别，显示识别结果。 |
| **总结评价** | 本节课基于face\_recognition项目完成了包含人脸检测、人脸关键点检测、人脸对比、人脸搜索以及人脸识别等方向的案例。 |