郑州大学毕业设计（论文）

题　　目：基于 PYNQ 的人脸识别设计与实现

指导教师：　　邓计才　　职称：教授

指导教师(校外)：　　曹健　　职称：副教授

学生姓名：　戴镇原　　学号：20142450401

专　业：　　电子信息工程

院（系）：　　信息工程学院

完成时间：　　2018 年 6 月 6 日

2018 年 6 月 6 日

基于 PYNQ 的人脸识别设计与实现

摘　要

近年来，人脸识别在支付、安保、机器人等领域得到了广泛应用，已成为计算机视觉领域的研究热点。人脸识别需要检测、对齐和识别等步骤，但 Haar 级联分类器、PCA 等传统方法在检测识别精度、鲁棒性等方面表现不佳。随着深度学习技术的发展，将深度学习算法应用到人脸识别，有效提高了人脸检测精度和鲁棒性。传统算法结合深度学习，已成为现在以及未来计算机视觉领域的发展趋势。

结合深度学习的人脸识别对算力要求较高，在 PC 机上开发算力相对足够，但是较难部署在嵌入式终端，需要综合考虑实时性、准确率、算力、功耗、成本、便携性、开发难易程度等因素。

本文针对上述问题，设计实现了一种基于 PYNQ 的人脸识别系统，在嵌入式终端实现结合深度学习的人脸识别——实时视频输入、人脸识别、显示结果输出。

课题主要工作有：1、调研人脸识别背景、意义、应用前景，确定课题目标；2、分析传统和结合深度学习的人脸识别算法特性，并在 PC 端实现，对结果进行验证、对比分析；3、提出基于 PYNQ 的嵌入式终端人脸识别系统的设计实现方案，并对系统各个部分设计实现原理进行详细分析；4、搭建嵌入式终端人脸识别系统，对课题方案进行测试验证与结果分析；5、对课题进行总结展望，提出系统改进优化方案。

本课题方案在实时性、准确率、功耗、成本等方面表现均较好。未来改进工作在算法方面人脸检测和识别可以尝试采用 YOLO、SSD、BNN 和 Resnet 等，实时性、准确性会更优。嵌入式终端实现方面若成本、功耗要求放宽，可以尝试 Nvidia 的 GPU 嵌入式开发平台—Jetson,性能应该会有较大提高且开发难度会降低。当然，仅使用 FPGA，开发难度虽大，但性能满足要求且成本低、功耗小，应当尝试突破。

本课题属于结合深度学习的计算机视觉算法的嵌入式实现这一研究方向，后续在此方向基于本课题将做更加深入的研究与实现。

关键词： 人脸识别，计算机视觉，深度学习，PYNQ，Movidius NCS

Design and Implementation of Face Recognition based on PYNQ

Abstract

In recent years, face recognition has been widely used in areas such as payment, security, and robotics, and has become a research hotspot in the field of computer vision. Face recognition requires steps such as detection, alignment, and recognition, but traditional methods such as Haar cascade classifiers, PCA do not perform well in detection accuracy and robustness. With the development of deep learning technology, deep learning algorithms are applied to face recognition, which effectively improves the accuracy and robustness of face detection. Traditional algorithms combined with deep learning have become the development trend in the field of computer vision now and in the future.

Face recognition combined with deep learning requires high computational power. It is relatively sufficient to develop computational power on a PC, but it is more difficult to deploy on embedded terminals . It requires comprehensive consideration of real-time performance，accuracy，computational power，power consumption，cost, portability, development difficulty and other factors.

This paper aims at the above problems, designs and implements a PYNQ-based face recognition system, which realizes face recognition combined with deep learning in embedded terminals—real-time video input, face recognition, and display result output.

The main tasks of the project are: 1. Investigate the background, significance, and application prospects of face recognition, and determine the goals of the project; 2. Analyze the characteristics of traditional face recognition algorithms and algorithms combined with deep learning.Implement them on the PC side, and verify and contrast the results. 3. Propose a design and implementation scheme of PYNQ-based embedded terminal face recognition system, and analyze the design and implementation principle of each part of the system in detail; 4. Set up an embedded terminal face recognition system to test and verify the project and analysis the results 5. A summary of the issues and outlook, proposed system improvement and optimization methods.

The scheme performs well in real time, accuracy, power consumption and cost. In the future, face detection and recognition in algorithms can be tried to use YOLO, SSD ,BNN ,Resnet and so on, and the real-time and accuracy will be better. If the cost and power consumption requirements of the development platform are not strict, Nvidia's GPU embedded development platform—Jetson, should be tried, and the performance should be improved greatly and the development will be easier. Of course, only using FPGA, the development is difficult, but the performance meets the requirements and the cost and power consumption is lower, we should try to break through.

This project belongs to the embedded realization of computer vision algorithm combined with deep learning, This will be followed by more in-depth research and implementation based on this project in this direction.

Key Words： Face recognition, computer vision, deep learning, PYNQ, Movidius NCS

目录

1 引言

1.1 课题背景|意义

计算机视觉在日常生活中的应用无处不在，其核心基础是图像的检测、识别、分割、特征点定位和序列学习等。由基础操作相互结合可以衍生出各种各样的技术，比如行人、车辆检测，图像风格迁移，图像生成，视频分割、预测等等，这些技术被广泛应用于手机移动端软件、机器人视觉、无人机、自动驾驶等领域。

在计算机视觉领域中，人脸识别一直以来都是研究的热点，其在身份识别等应用上相比其他方法有得天独厚的优势。

身份识别技术在当今社会越显重要，互联网高速的发展，信息安全的加强是需要重视的问题。身份识别广泛用于便捷支付、安保、安检、人员统计等领域。身份识别方法有很多，如密码、口令、身份证、工作证等，虽然这些方法较完备，但是这些方法在识别的时候给使用者额外增加了区分信息，使用不便，且附加信息容易丢失，安全性不高。传统身份识别方法不太能够适应现在社会的需要。

有别于传统身份识别技术，生物识别技术是利用生物个体本身的特征来进行识别，比如利用人脸、语音、指纹、虹膜等进行识别。这些特征往往具有唯一性，且相对于传统方法更加可靠、方便，因此逐渐替代传统身份识别方法。

在常用的生物身份识别技术中，人脸识别又是使用最为广泛的技术，因为其更加方便、快捷、友好、易于交互。

此外，人脸识别技术也常用于摄像中的自动对焦到人脸，自拍软件中的实时脸部贴图，机器人视觉中的表情分析、人机交互等等，同时人脸识别也和其他相关技术结合，使其应用更加广泛。

传统计算机视觉已有十分坚实的理论基础，但其在精度上似乎遇到了瓶颈，无法达到更好的效果。随着深度学习兴起，其在图像识别、语音识别、自然语言处理上的应用收到了很好的效果。在计算机视觉上应用深度学习，其准确率是传统方法几乎难以企及的。深度学习不需要人为根据先验知识去指定特征，然后运用特定的提取方法结合分类器进行识别，而是深度神经网络像一个黑盒子，将大量人脸数据灌入，通过训练，即可得到准确率大大超过传统方法的模型[13]，这是十分理想的，结合深度学习的计算机视觉也是现在和未来的计算机视觉领域的发展趋势。

算法上由于深度学习的加入，精度上大幅度提高，但算法最终想要发挥最大功用，真正应用到人们的日常生活中，就要进行算法的终端实现。算法精度高了，这也给终端实现造成了压力，因为深度神经网络的运用需要进行大量的

数据运算，对终端实现平台的算力有较高要求，同时还要兼顾考虑功耗、成本、便携性等因素，达到平衡。

目前人脸识别终端实现方案大多采用 ARM、DSP、GPU、ASIC 等，ARM、DSP 的算力十分有限，难以达到实时、准确的要求，若结合云端服务器确实可以提高其准确性，但是需要联网且稳定性、实时性也难以保证；GPU 算力足够但功耗大、成本高；ASIC 为专用芯片，通用、灵活性不强。FPGA 是介于 GPU 和 ASIC 之间的一种开发平台，其适合做量大、重复性强、结构相对简单的运算，但对于结构较为复杂的算法，FPGA 开发难度较大，很难完整实现。综上，这就需要从多方面综合考虑，探索更合适的嵌入式终端实现方案。

人工智能-深度学习-计算机视觉算法的嵌入式终端实现这一领域的研究与实践，对推动人工智能的发展、应用、推广，都有着深远意义。

1.2 课题目的

1．研究、学习人脸识别原理、方法，并加以实现。对比传统方法和结合深度学习的方法的异同，从实时性、精确度等方面考虑，选取最合适的人脸识别算法方案进行改进、运用；

2．熟悉人脸识别不同的嵌入式终端实现方案，对比其优劣。并针对其不足设计改进，实现有效性、可靠性、功耗、成本、便携性等相对均衡的方案；

3．克服传统人脸识别在算法和硬件实现上的不足，比如：算法上仅单纯使用基于知识或基于统计的方法，准确度难以保证；硬件实现上仅使用 ARM 或 DSP 等难以保证实时性而利用 ARM/DSP+FPGA 构造上、下位机系统又过于复杂、成本过高、通信难以保证等；

4．基于 PYNQ（结合 Movidius NCS）设计并实现一套人脸识别终端系统，能够实时、准确进行人脸识别应用；

5．在 PYNQ 上（结合 Movidius NCS）使用 Python(可结合 C/C++)进行开发，利用 Python 丰富的可利用资源库和 PYNQ-Z1 的 ARM+FPGA 架构特性以及 Movidius NCS 对神经网络的加速，低功耗、高速的在嵌入式终端实现结合深度学习的人脸识别；

6．深入研究这种低功耗、高速、成本可接受、体积合适的人工智能终端设备的系统设计与开发。

1.3 课题安排

本课题论文共六章，首章为引言，总体介绍计算机视觉—人脸识别的算法、嵌入式终端实现和应用、发展趋势，以及课题目的；第二章为人脸识别算法分析，对人脸识别领域传统和结合深度学习的算法进行分析、对比；第三章为本课题人脸识别系统的设计与实现方案，详细介绍系统和各模块设计实现原理；

第四章为验证与分析，得出课题结果并进行分析对比；第五章为课题总结以及针对本课题相关工作的改进方案与未来展望。

2 人脸识别算法

2.1 人脸识别算法综述

人脸识别[10]总体包括以下几部分：图像采集、图像预处理、人脸检测、特征点定位|人脸对齐、人脸识别等，如图 2-1 所示。

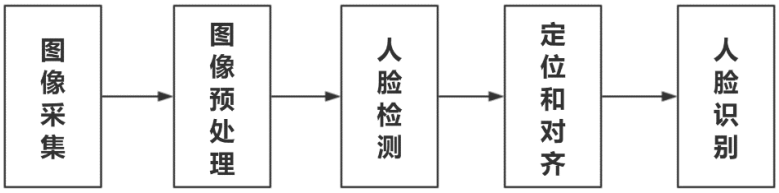


图 2-1　人脸识别流程

2.2 传统人脸识别算法

2.2.1 图像采集

利用摄像头、HDMI（嵌入式设备）从外部输入或直接从内存中读取图像、视频，采集的图像用于后面的处理和识别。要在 PC 机调用摄像头、或从内存获取图像、视频可直接利用 OpenCV 或其他工具的特定函数库较为方便的实现。

2.2.2 图像预处理

获取的人脸图像由于环境、随机噪声等多种因素影响，输入的原始图像往往存在较大随机干扰，导致其特征不明显，若不对其进行处理，将大大影响后续的检测识别精确度。常用的人脸图像预处理方法有图像转正、增强、滤波、锐化、灰度化、二值化、归一化等。通过这些图像预处理操作，最大程度的保留必要信息，去除干扰信息，为后续检测、识别做准备。

人脸图像转正是对偏斜的人脸图像进行矫正，增大检测可能性;人脸图像增强是改善图像质量，使图像更加清晰且更易于计算机的处理与操作，像常用的滤波如：中值滤波、均值滤波等，可以有效地抑制图像中的噪声；人脸锐化则可以强化、突出人脸边缘信息，也称边缘增强；灰度化、二值化则是将彩色图像转化为单通道灰度图像并进行二值化操作，这样可以去除背景影响；归一化是将灰度值统一限定在同一标准范围，削弱光照条件不同造成的影响，提高检测适应性。

经过图像预处理，图像变得相对较为“干净”，后续进行画面中人脸的检测。

2.2.3 人脸检测

人脸检测就是输入图片或视频信号，在其中找出单个或多个人，并返回其坐标和范围。传统人脸检测是利用特征加分类器，尤其是 Viola 和 Jones 提出的开创性算法：Haar 特征＋AdaBoost 级联分类器[13]，其达到很好的实时检测性。

但只适合在约束环境条件下如：人脸数单一、背景相对简单、人脸较正立、光线影响不大等情况下使用擦能达到符合要求的精确度。在非约束环境条件下，如：多尺度、数量大、遮挡、倾斜、光照条件恶劣、分辨率低等情况下，即使可以通过图像预处理对其进行改善，但该算法的精确度仍难以保证。

OpenCV 中有预先训练好的 Haar 级联分类器，用于人脸、眼、鼻等器官检测，可直接调用完成限定约束环境下要求不高的人脸及器官检测，如图 2-2 所示。该分类器可以实时检测，但是对于侧脸和器官的检测精确度实在不高，这也与训练时提供的数据有关。开发者也可以自行提供数据训练相关模型，有助于提高检测精度。

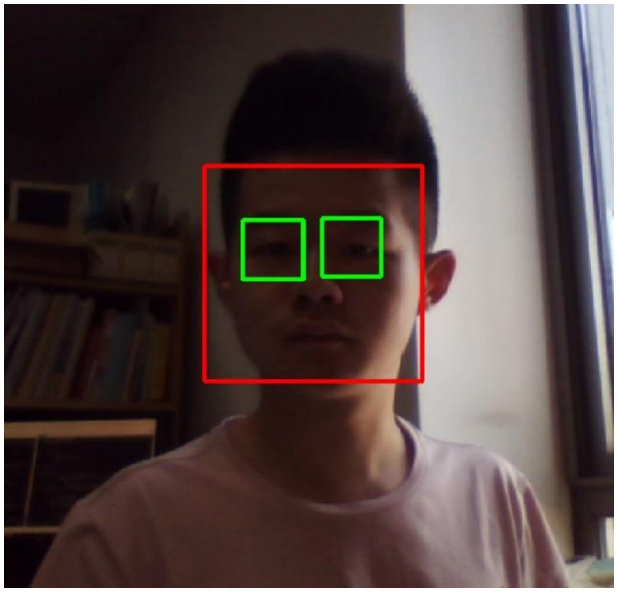


图 2-2 Haar 级联分类器—人脸检测

使用传统的 Haar 级联分类器在非约束测试集 FDDB 中进行测试时显示：误检数限定为 10 个和 500 个时，检测准确率仅为 10%和 52.8%。要想用于非约束环境条件下如大规模的网络安防布控等，还需要大幅度提高算法的鲁棒性和精确度。

人脸检测可以检测到画面中的人脸并进行位置标记，可用于人流量统计也是后续识别的基础。

2.2.4 特征点定位|对齐

人脸特征点定位|对齐即输入人脸图像，确定并标记出人脸的关键点位置，如人脸轮廓、眉毛、眼睛、鼻子、嘴巴，如图 2-3 所示。

比较传统的人脸对齐算法有 ASM、AAM 等，目前比较流行的有 RSR、LBF 等。ASM 较容易实现，适用于正脸。AAM 使用全局纹理信息，精度更高，但对光照条件和多姿态变化适应性不好。CLM 则综合二者特性，性能得到较大提高。

人脸对齐应用很多，如器官定位、跟踪，活体检测、表情识别，虚拟|增强

现实，年龄|性别估计等等，同时也是后续人脸识别的必要基础。

图 2-3　特征点定位|对齐

2.2.5 人脸识别

特征提取

在之前人脸检测、对齐获取的信息基础上，进行人脸特征提取。常用特征提取方法包括基于全局和基于局部的特征提取。

基于全局的特征提取是用特定方法建立子空间，并将人脸图像（待检测和目标）映射到子空间，在子空间得到其特征值。常用的有 PCA[9]和 LDA 以及其变

形算法。此外还有利用空间域信息的方法如：DFT、DCT 等，它们从频域提取图像特征，不需要运算，且可以利用 FFT 加速实现，因此应用较为广泛。但基于全局的特征提取较易受到人脸局部变化的影响，如在表情变化、遮挡等情况下会影响识别结果。

基于局部的特征提取将人脸拆解用局部信息来做特征提取，这样就能克服局部变化造成的影响。常用的方法有弹性图匹配、局部二值模式、HOG 特征等方法。

HOG（边缘梯度直方图）特征利用边缘梯度可以很好的描述物体局部外观和形状，基于 HOG 特征训练的 SVM 运用在人脸检测上，收到比 Haar 级联分类器更好的效果，检测率更高，如图 2-4 所示。（在图 2-3 人脸对齐中，人脸检测就用的是 HOG-SVM）

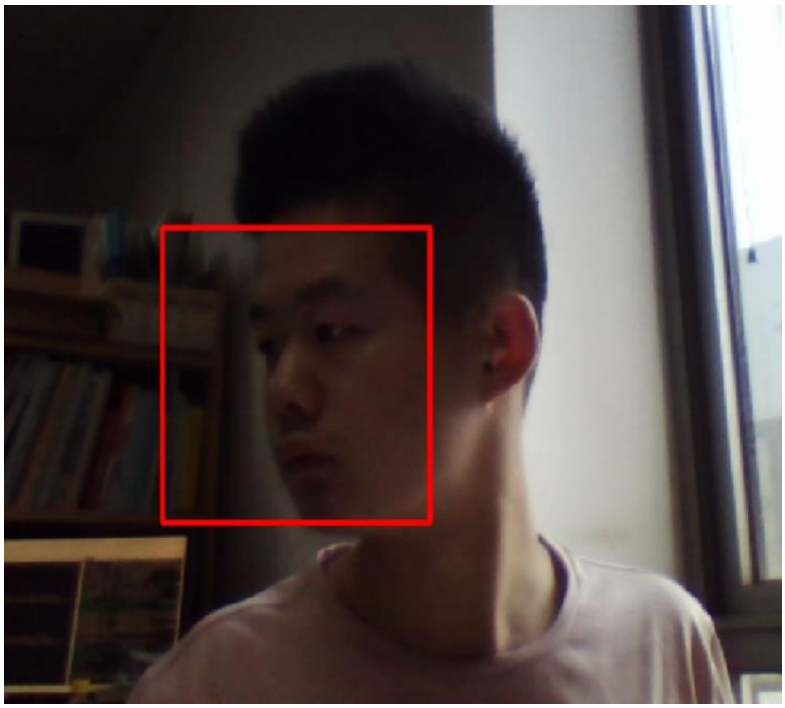


图 2-4 HOG-SVM—人脸检测

以上特征均为几何特征，实际运用过程中常将基于全局和基于局部的特征提取结合使用，且同时使用统计特征来达到更好的效果。

人脸匹配

将提取的人脸特征利用设计训练好的模型进行比对（比如计算特征间欧氏距离），输出其相似度（如欧氏距离大小，越小越相近）与结果。在此基础上结合相关策略可以实现人脸验证、人脸识别、人脸检索、人脸聚类等算法。

人脸验证是 1:1 进行对比，看是否是同一人；人脸识别和人脸检索是 1:N、N:N，在数据库中找到当前目标；人脸聚类是 N:N,在待检测图像中将同一人的图像归类整理。

不同人脸算法常应用在不同场合，如人脸验证常用在车站安检进行本人和身份证比对；人脸识别和检索常用与门禁和侦查；人脸聚类常用于智能相册，管理图片，将相同特征的相片归类整理。

2.3 结合深度学习的人脸识别算法

前述传统人脸识别算法总体来说是特征+分类器，即根据提取的特征进行分类。在较为复杂情况下，这些人为提取的特征往往不够全面、适应性不强，难以准确描述不同情况下人脸的信息，因此在复杂多变情况下人脸识别准确率不高。

而近年来兴起的深度学习可以很好的解决这个问题，开发者只需要向深度神经网络提供足够多的数据，神经网络就可以用过训练，学习到输入数据的特征信息，比如人脸的特征信息，进而进行人脸检测、识别等任务。

深度学习中的 CNN（卷积神经网络）非常适合应用于计算机视觉领域，在图像处理上的应用取得了传统算法难以达到的精确度，同时随着近年来的优化，各种网络结构越来越精巧、简练，在精确度保证的同时也达到了较高的检测速度，可以满足在移动端的实时检测需求。

人脸是深度学习——目标检测、识别等领域研究较多的方向，人脸识别包含了目标的检测、定位、识别等子任务。

同样，必要的图像预处理后，进行人脸检测与识别。深度学习下的人脸检测[13]常用 CNN、RCNN、Fast-RCNN[1,2]、Faster-RCNN、Yolo、SSD 等，人脸识别常用 VGGNet、GoogleNet、ResNet、FaceNet 等。这些典型神经网络已经过研究者大量实验验证，在相应检测、识别任务中都收到令人满意的效果。同时，不同网络也有各自的特点、优势和不足。

典型 CNN 包含的层有：输入层、卷积层、激励层、池化层、全连接层、输出层以及其它层如归一化层、切分层、融合层等。

输入层：数据输入;

卷积层：用卷积核进行特征提取和映射;

激励层：非线性映射（卷积是线性运算，只有非线性化，才能表示更复杂的函数，网络效果才能更好）;

池化层：下采样，特征稀疏，减少数据运算量;

全连接：CNN 尾部，重新拟合，减少特征信息损失;

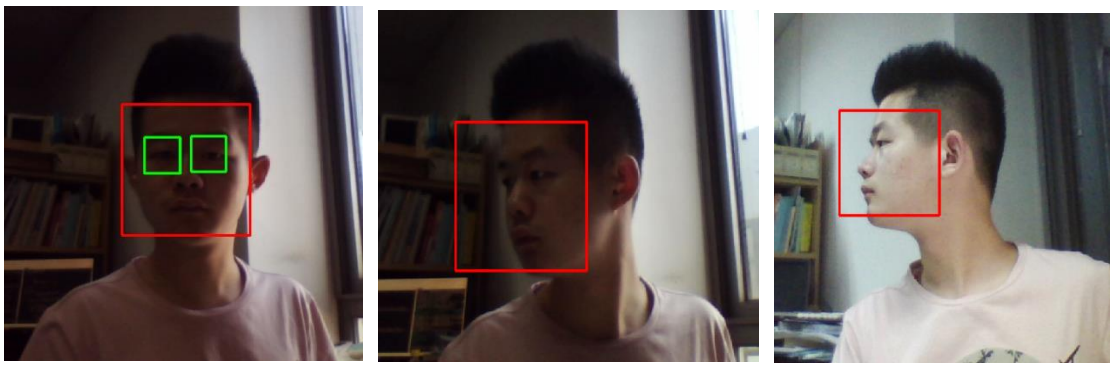
输出层：输出结果;

其它层：归一化层（特征归一化，防止“梯度弥散”）、切分层（分区域

学习）、融合层（将分区域单独学习的分枝进行融合）；

那些典型的深度神经网络，正是将这些基础层进行有规律的结合而来。

一般CNN用于人脸检测如图2-5所示，将常用人脸检测算法效果进行对比。虽然 CNN 在检测精度上大幅度提高，但是由于数据计算量较大，检测速度较低，难以达到实时性需求，在移动端终端更是难以应用。随后的由 RCNN 改进的 Fast-RCNN 和 Faster-RCNN 则在检测速度上得到了较大改善，Faster-RCNN 已可以实现对视频信号的实时人脸目标检测。Yolo 相对于 Faster-RCNN 省去了其检测过程中的预检测 RPN，这使得检测速度进一步提高，在移动终端设备上已可以达到较理想的实时检测效果，但是精确度有所下降。SSD 则综合了 Faster-RCNN 和 Yolo 的优势,在精确度和检测速度上均有较大改善。



Haar 级联分类器　　HOG-SVM　　CNN

图 2-5　不同算法人脸检测对比

人脸识别常采用 VGGNet、GoogleNet、ResNet、FaceNet 等网络。

VGGNet 常用于目标检测分类，比如 Cifar 数据集训练的 VGGNet 在特定种类目标识别时精度很高。VGGNet 结构如图 2-6 所示

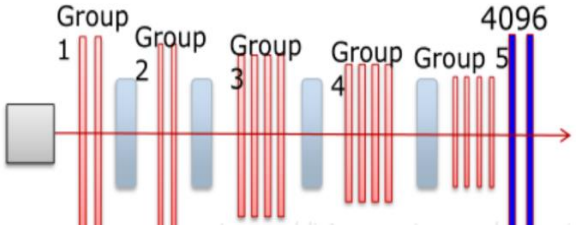


图 2-6 VGGNet 结构图

GoogleNet 的特点在于利用不同分辨率的卷积核对图像特征进行提取，最后把不同卷积核提取的特征图进行融合，可较大提高其检测效率和精度。GoogleNet 实际由 Inception 结构堆叠而成，如图 2-7 所示。

Resnet(残差网络)，结构如图 2-8 所示。它将低层次学习特征和高层次学习特征进行融合，这样反向传播就不会因为网络层次过深而引起“梯度弥散”而使训练低效，极大提高模型训练的效率，在训练效率上优于其他网络，同时也为更深层次模型构建提供可能。

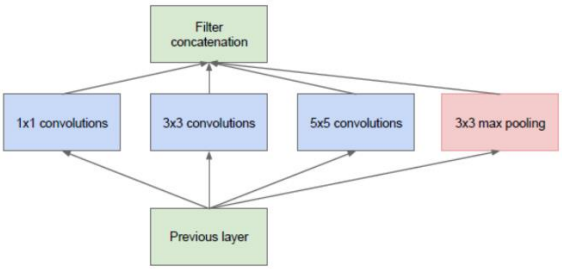


图 2-7 Inception Module

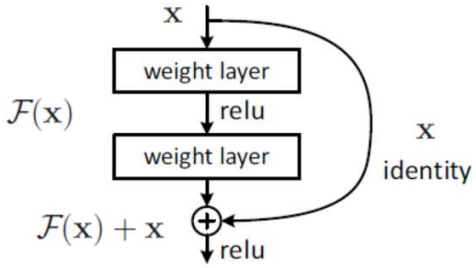
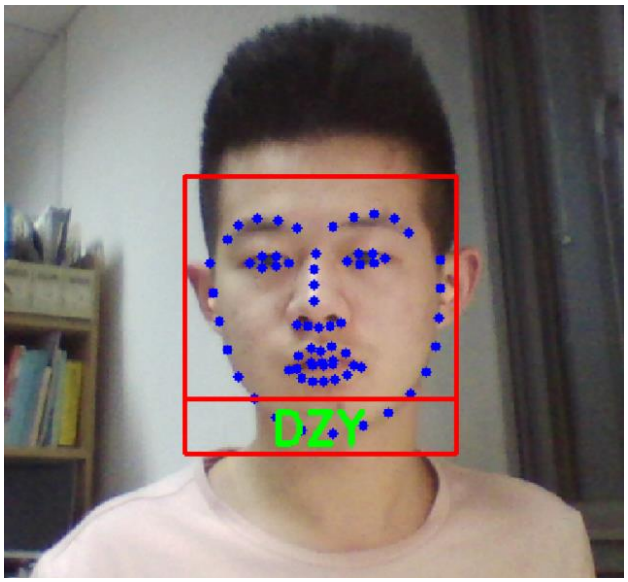


图 2-8 ResNet 关键操作

在不同场合下，需要合理选择适当的网络运用到人脸识别中，若对实时性要求很高，则可使用 Yolo 来进行人脸检测；若对精确度要求较高，则可选用较深层次的 Resnet 来进行人脸识别，如图 2-9 所示。在嵌入式终端设备上更适合采用 Faster-RCNN、Yolo 等实时性较好的模型来实现。

2.4 本章小结

本章对传统和结合深度学习的人脸识别算法进行了分析，对比了其异同和特点。总体来说传统识别算法在精度上远不如结合深度学习的识别算法，但深度学习需要大量数据来进行模型训练，对数据、算法、算力都有较高要求。实际运用过程中常常二者相互结合使用，比如用 HOG-SVM 进行人脸检测，用 RestNet 做人脸识别，这样在有效性和可靠性上都可以达到较好效果。



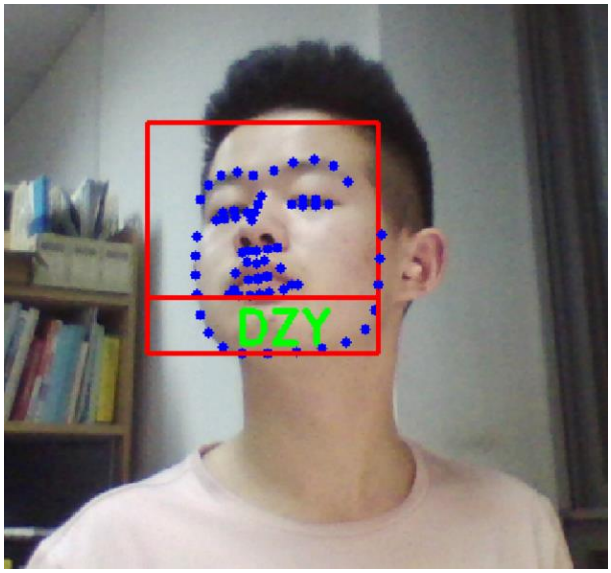


图 2-9 ResNet—人脸识别

3 设计与实现

3.1 系统设计

人脸识别系统主要由图像采集、主控、结果输出等几部分构成。图像采集部分使用 USB 工业高清摄像头进行实时视频信号的输入；主控采用 PYNQ[3,4]结合 Movidius NCS 神经网络加速棒通过深度神经网络来进行人脸识别；结果输出使用 HDMI 将实时视频信号输出至移动式显示屏显示识别结果，系统结构如图 3-1 所示。

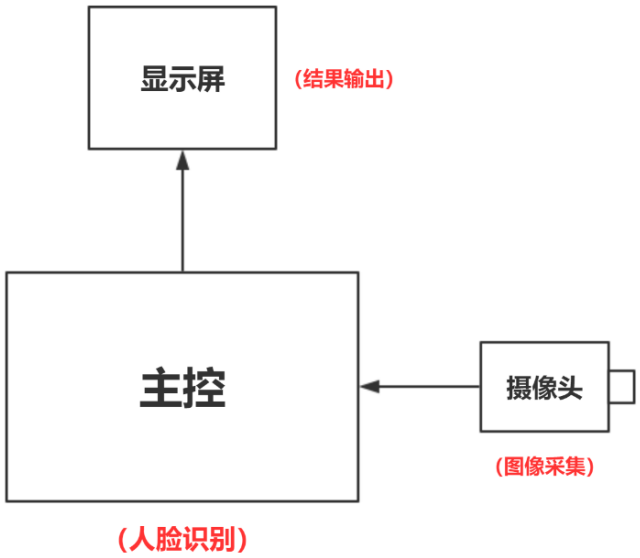


图 3-1 人脸识别系统框图

系统识别前，需要进行人脸录入，本系统仅需录入识别目标的单张照片即可达到较理想的识别检测率。录入待识别人脸后，待检测者出现在摄像头前时，摄像头捕捉到人脸画面后输入至主控进行人脸识别，识别结果实时输出至显示屏显示。当识别到录入的人脸时，画面被绿框框出，并对应显示出目标 ID；当识别到未录入人脸时，画面被红框框处，系统运行流程如图 3-2 所示。整体操作较为简便，实时性、可靠性均较好，单张录入，在比较复杂的环境下仍能保证较高检测率。

软件开发方面，主要在 ARM 的嵌入式 Linux 上顶层采用 Python 进行编程开发，同时需要结合 Block Design、C++以及 Verilog 进行 FPGA 的 IP 核设计、bitstream 综合等。

FPGA 和 NCS 配置完成后，顶层在 ARM 上通过 Jupyter 网络服务器和 IPython 解释器可以使用 Python 进行调用，同时结合 Python 丰富的支持库如 OpenCV、tensorflow、caffe 等，以完成实时人脸识别任务。

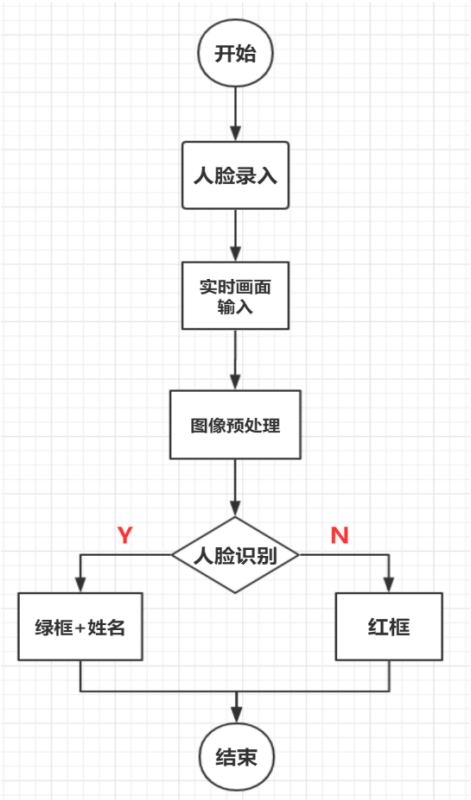
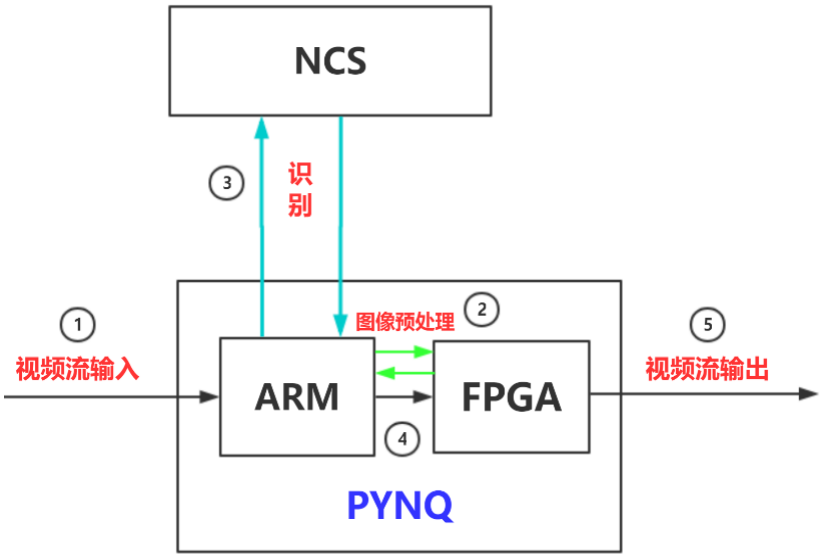


图 3-2　人脸识别系统功能流程图

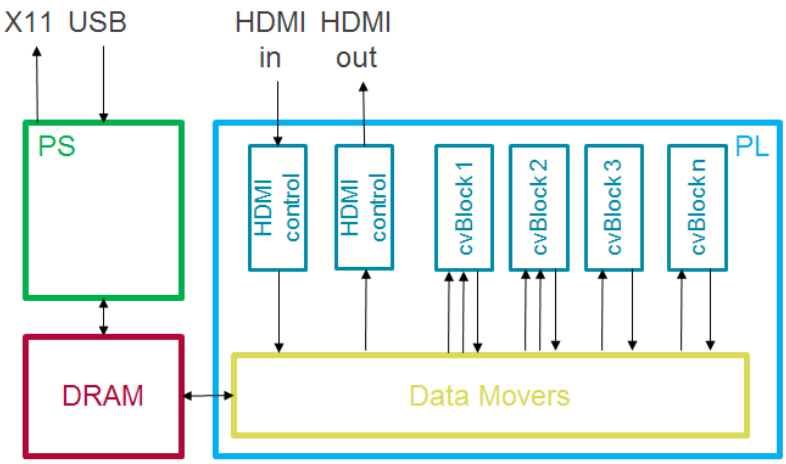
3.2 模块设计

3.2.1 主控

主控包括 PYNQ 和 NCS，总体逻辑为 PYNQ-ARM 负责调度和视频流输入、PYNQ-FPGA 负责图像预处理和视频流输出、NCS 负责识别。在 ARM 调度下，视频流输入、预处理、识别、输出有序、实时进行。如图 3-3 所示（a 图中 1、2、3、4、5 为执行顺序）



a — PYNQ + NCS



b — PYNQ

图 3-3 人脸识别系统主控逻辑图

PYNQ

本课题核心处理板选择 Xilinx 的 PYNQ[8]，如图 3-4 所示。PYNQ 主要芯片采用 ZYNQ XC7Z020，其内部异构双核 ARM-A9 CPU+FPGA，PS（ARM）和 PL（FPGA）之间通过片上高速 AXI 总线进行数据交互，有效克服 ARM 和 FPGA 间数据传输的低速、延时、外界干扰等问题，可以充分结合二者特性，联合开发。

在 PYNQ 的 ARM-A9 CPU 上运行的软件包括 Jupyter Notebooks 网络服务器、IPython 内核和程序包、Linux-Ubuntu 以及 FPGA 的基本硬件库和 API。

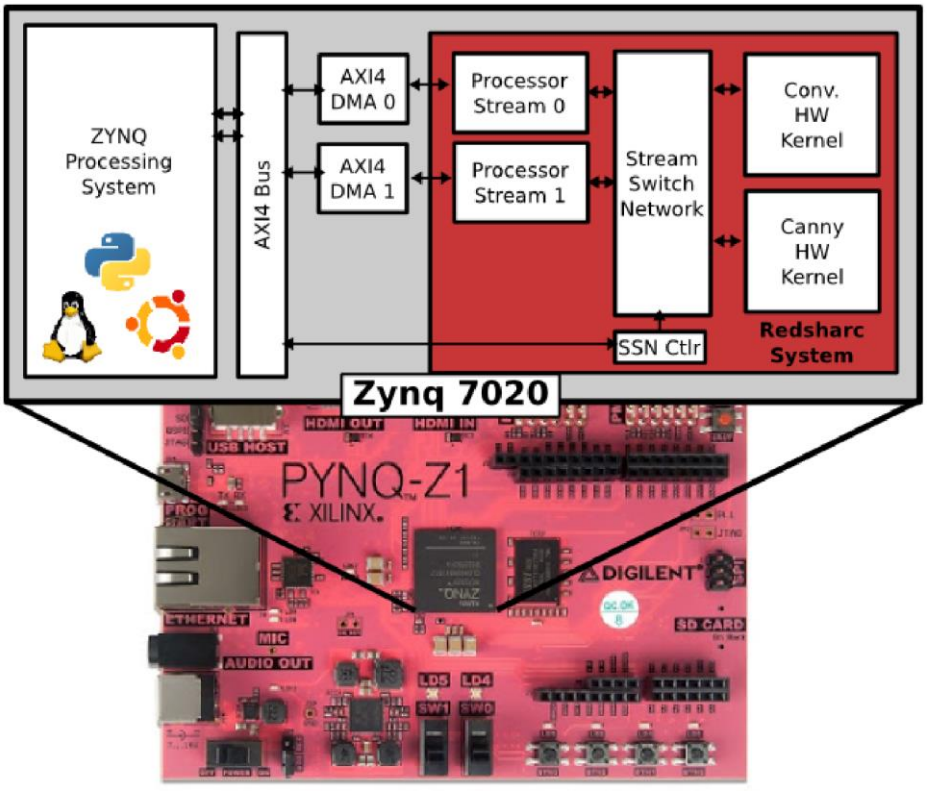


图 3-4 PYNQ 内部结构

在 Linux 系统下，通过 Jupyter Notebooks 网络服务器访问 PYNQ，在 IPython 内核和程序包支持下,可以在 Jupyter Notebook 中用 Python 进行 FPGA 等资源的调用、开发。如图 3-5 所示。

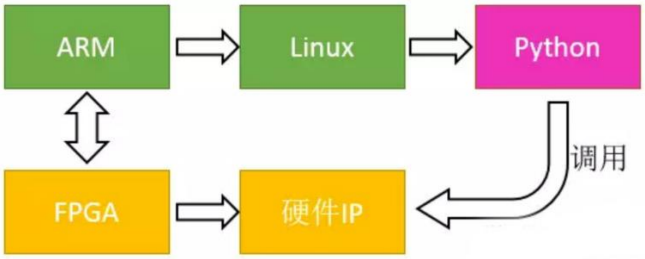


图 3-5 PYNQ 使用-逻辑导图

PYNQ 开发，需要配置系统，在 PC 机上向 SD 卡中写入系统镜像文件，然后将 SD 卡插入 PYNQ 卡槽中，设置为 SD 卡启动、外部直流或 USB 供电（实验中发现外部直流供电时板子性能更好）、连接板子以太网口至路由器或 PC 机(需要必要的网络配置)，接通电源，PYNQ 正常启动后开始进行开发。

PYNQ 启动配置步骤，如图 3-6 所示。1、2、3、4、5、6 分别为设置系统启动方式、设置供电方式、插入 SD 卡、USB 供电、以太网口、电源开关。

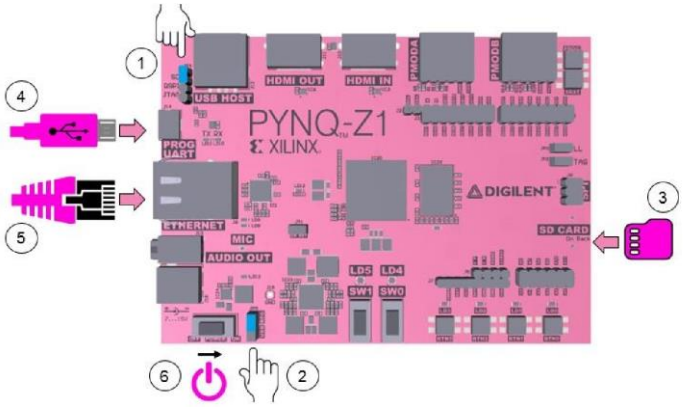


图 3-6 PYNQ 启动配置示意图

PYNQ 启动后在 PC 机上通过 chrome 浏览器访问板载网络服务器，进入 Jupyter Notebook 编译环境，如图 3-7 所示。在 Jupyter 中使用 Python 编程，调用硬件库及加载 Overday 来对硬件平台进行控制和交互。

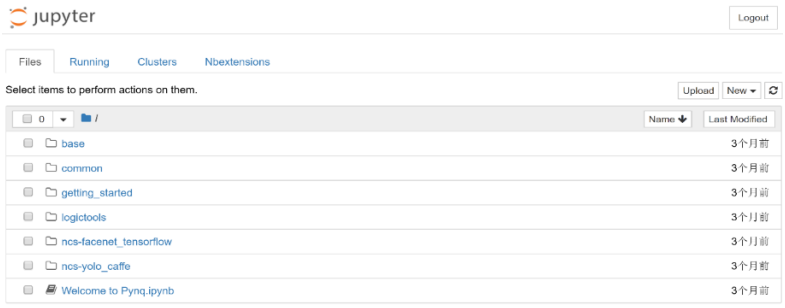
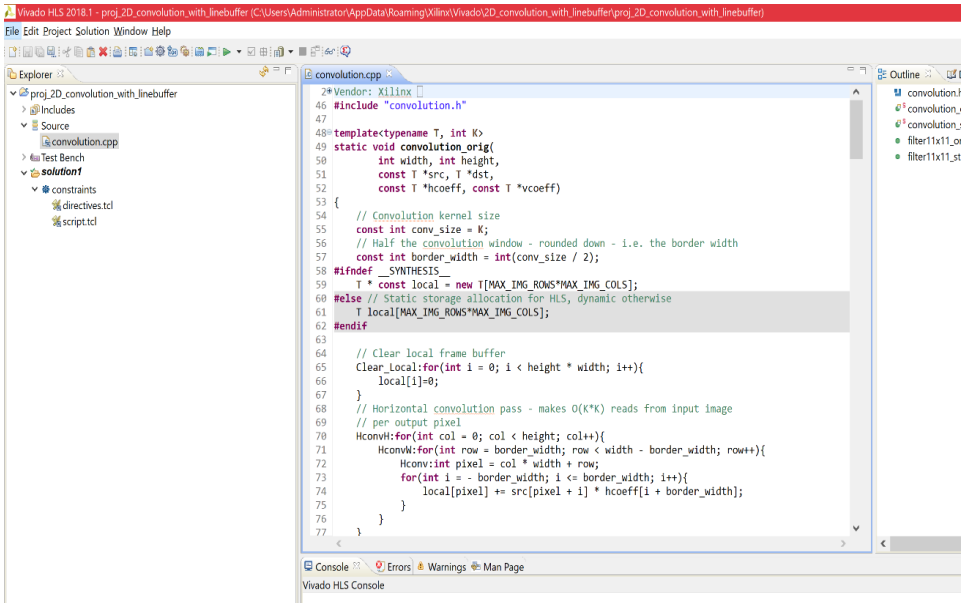


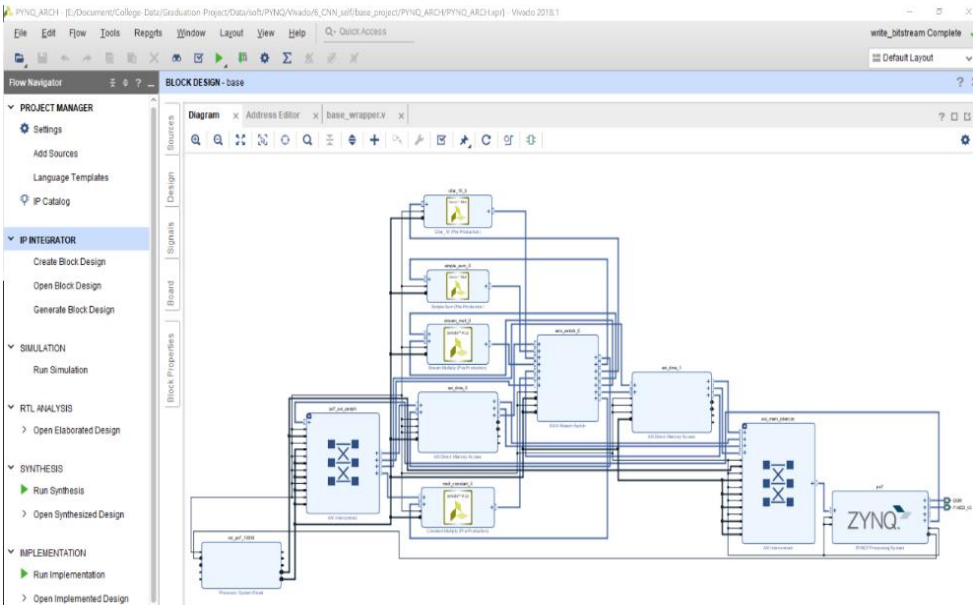
图 3-7 Jupyter Notebook

PYNQ 最大的特色在于，可以在顶层使用 Python 开发并调用硬件部分（FPGA）进行相关处理、运算的加速，比如图像预处理、神经网络等。在 ARM 上配置好启动环境后，需要根据具体的算法来设计 FPGA 部分，最终生成配置 FPGA 部分逻辑电路的 bitstream 文件，顶层调用.bit 文件部署 FPGA 后就可以实现算法硬件加速。

FPGA 设计要用到 Vivado HLS 和 Vivado，在 Vivado HLS 或 Vivado 中利用高级语言 C++或 Block Design 设计针对具体算法的 IP 核，然后在 Vivado 中利用 Block Design 或 Verilog 调用 IP 来进行 FPGA 整体设计、综合，生成 bitstream（.bit）后导入 PYNQ 调用使用。设计演示如图 3-8 所示。



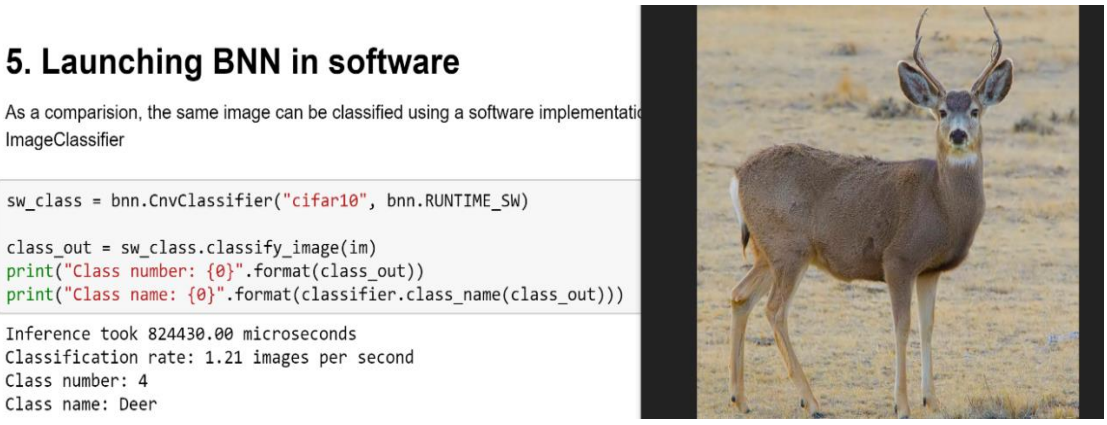
HLS—IP 设计



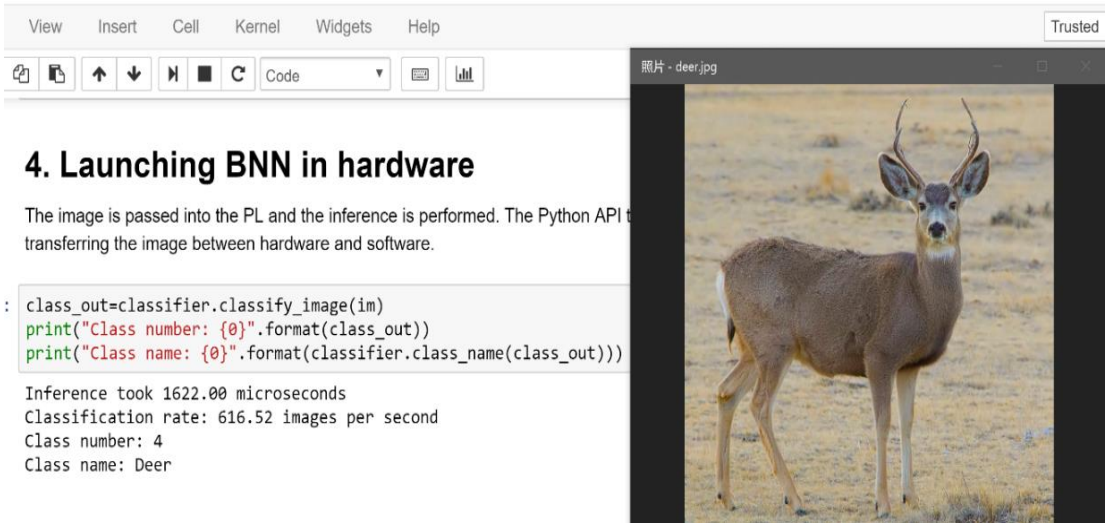
Vivado—bitstream 综合

图 3-8 PYNQ—FPGA 设计演示

在 PYNQ 的 FPGA 加速下，图像处理速度相比在 ARM 上要快很多，如在目标识别任务中，如图 3-9 所示,利用 VGGNet[5,6]识别特定物体，ARM 处理速度为 1.21fps，FPGA 加速下处理速度为 616.52fps，FPGA 加速下极大提高识别速度可以实现实时目标识别。



PYNQ—ARM



PYNQ—FPGA 加速

图 3-9 PYNQ 目标识别

在人脸识别任务中，当图像预处理在 PYNQ-FPGA 加速下执行时要比在 PYNQ-ARM 上执行快的多，这使系统识别速度有较大提升，后面将详细介绍。

Movidius NCS

Movidius NCS 是 Movidius 新推出的基于 Myriad 2 视觉处理单元的神经网络加速棒。其最大特性在于低功耗下可以提供 1000 亿次/秒的浮点运算，可以集成在嵌入式终端设备上，使终端设备有能力直接运行实时深度神经网络，为人工智能、深度学习应用离线部署提供支持。

深度神经网络运算量较大，NCS 在使用过程中会有发热现象，因此其在原型机上包裹一层铝制外壳，增强散热效果。同时在使用时需要由带电源的 USB

集线器单独供电（I>500mA），仅由板载 USB 供电无法启动。NCS 如图 3-10 所示。

使用 NCS 进行开发，需要用到官方提供的 SDK，其中包含 Tools 和 API 两部分。在 PC 机上配置完整 SDK（Tools+API），在嵌入式控制板（PYNQ 等）上仅配置 SDK 的 API，配置完成后，开始 NCS 深度神经网络的开发。

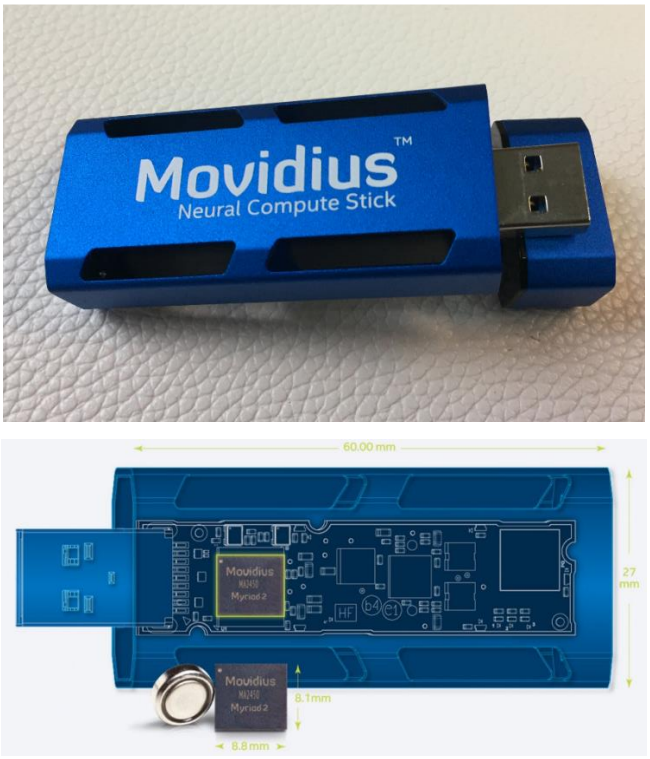


图 3-10 Movidius NCS

大致开发流程分为三步：

1、模型搭建、训练。在 PC 机或服务器上利用 tensorflow 或 caffe 框架进行神经网络的搭建、训练、优化，并保存生成的模型为 tensorflow 或 caffe 模型标准形式。（目前 NCS 仅支持 tensorflow 和 caffe）

2、模型编译、转化。标准 tensorflow 和 caffe 模型无法被 NCS 直接使用，需要在 PC 机上针对标准模型利用 SDK—Tools 中的 compiler(编译器)将其准换为 NCS 专用格式的模型文件（.graph），然后导入嵌入式控制板中。

3、模型调用、推理。在嵌入式控制板上，利用 SDK 的 API 调用对应的模型（.graph）进行推理，实现相应神经网络功能。

NCS 的使用，可以极大加速神经网络的推理速度，在低功耗的嵌入式终端可以起到明显作用。如利用 YOLO 进行目标检测、识别，在 ARM 上单帧图片识别就需要花费将近 20 秒的时间（即 0.05fps）；而在 NCS 加速下，图像识别速率提升至 2.9fps,采用其它方法改善后速率可达到 6fps,基本满足实时目标检测、识别的需求。

如图 3-11，识别出了画面中的不同物体，准确率、速度均达到要求。

值得一提的是，NCS 加速棒支持叠加，类似 GPU。根据任务需要，可以采用多根加速棒叠加来运行单个或多个模型。通过合理分配不同网络模型至多个加速棒，可以在嵌入式终端实现多功能、更复杂的深度神经网络，并保证其实时性、低功耗。

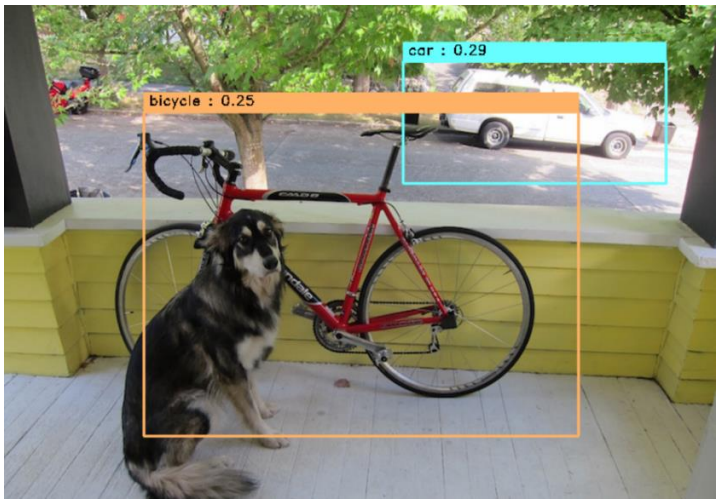


图 3-11 NCS-YOLO-目标识别

3.2.2 图像输入、输出

图像输入，工业高清摄像头接至主控板 PYNQ 的 USB 接口，在 PYNQ-ARM 的 Linux 上配置 OpenCV 环境,利用 OpenCV 库函数 cv2.VideoCapture()由 ARM 驱动 USB 接口进行图像采集，此功能函数使用较为灵活,参数可以为内存中图片或视频的路径，则从内存中读取相应的图片或视频；若为整数，则为连接在 USB 接口的摄像头编号，从 0 开始，可以选择需要调用的摄像头。图像输入时限制分辨率为 640*480，以 BGR 通道顺序输入。

实际应用时，输入图像每隔 2 帧处理一次，在不影响检测准确率的前提下，释放多余占用资源。如图 3-12 所示。

```
# Webcam resolution
frame_in_w = 640
frame_in_h = 480

# Configure webcam - note that output images will be BGR
#videoIn = cv2.VideoCapture('/home/xilinx/jupyter_notebooks/ncs-facenet_tensorflow/images/2.jpg')
#videoIn = cv2.VideoCapture('C:\\Users\\Administrator\\face_detect\\face_detect.mp4')
videoIn = cv2.VideoCapture(0)
videoIn.set(cv2.CAP_PROP_FRAME_WIDTH, frame_in_w);
videoIn.set(cv2.CAP_PROP_FRAME_HEIGHT, frame_in_h);

print("Capture device is open: " + str(videoIn.isOpened()))
```

```
# 读入一帧图像
ret, frame = videoIn.read()
i = i + 1
# 每隔2帧处理1帧图像
if(i > 2):
    i = 0
```

图 3-12 图像输入

图像输出，由 HDMI 输出处理后的视频信号至显示屏实时显示结果。HDMI 由 ARM 调用 FPGA 硬件库实现。先加载 bitstream(.bit)文件，部署 FPGA 内部逻辑资源，然后导入 PYNQ 的 python 视频信号处理程序包，初始化 HDMI 输出的分辨率、通道等，配置为和输入相同，分辨率 640*480，通道顺序为 BGR。之后开启 HDMI，准备输出。如图 3-13 所示

```
from pynq.overlays.base import BaseOverlay
from pynq.lib.video import *

# Load the base overlay
base = BaseOverlay("base.bit")

hdmi_out = base.video.hdmi_out

# Configure the HDMI output to the same resolution as the webcam input
mode = VideoMode(frame_in_w,frame_in_h,24)
hdmi_out.configure(mode, PIXEL_BGR)

# Start the HDMI output
hdmi_out.start()
```

图 3-13　图像输出

3.2.3 图像预处理

图像由 USB 摄像头输入后，在 PYNQ 上进行图像预处理。（实验表明，在 PYNQ-FPGA 加速下进行图像预处理比在 ARM 上处理速度提升 3 倍以上，加速效果明显，较大程度提高系统识别实时性）图像预处理包括图像尺寸、通道顺序、图像格式标准化的调整，以满足人脸识别网络的要求。输入是分辨率为 640*480，通道顺序为 BGR 的彩色图像，人脸识别网络要求分辨率为 160*160，通道顺序为 RGB，故作相应调整。图像尺寸调整采用 cv2.resize()函数，图像通道顺序变更采用 cv2.cvtColor(preprocessed_image,cv2.COLOR_BGR2RGB)函数，图像格式标准化调整采用 whiten_image(source_image)模块。图像预处理（PYNQ-ARM、PYNQ-FPGA）如图 3-14 所示。

```
def preprocess_image(src):
    # scale the image
    NETWORK_WIDTH = 160
    NETWORK_HEIGHT = 160
    preprocessed_image = cv2.resize(src, (NETWORK_WIDTH, NETWORK_HEIGHT))

    #convert to RGB
    preprocessed_image = cv2.cvtColor(preprocessed_image, cv2.COLOR_BGR2RGB)

    #whiten
    preprocessed_image = whiten_image(preprocessed_image)

    # return the preprocessed image
    return preprocessed_image
```

PYNQ-ARM

```
# 图像预处理(FPGA)——大小、通道顺序、格式调整
def preprocess_image(src):

    # 大小调整
    NETWORK_WIDTH = 160
    NETWORK_HEIGHT = 160
    # resize
    xv2.resize(src, preprocessed_image)
    # convert: BGR—>RGB
    xv2.cvtColor(preprocessed_image)
    # 格式调整
    preprocessed_image = whiten_image(preprocessed_image)
    # 返回预处理后的图像
    return preprocessed_image
```

PYNQ-FPGA

图 3-14　图像预处理

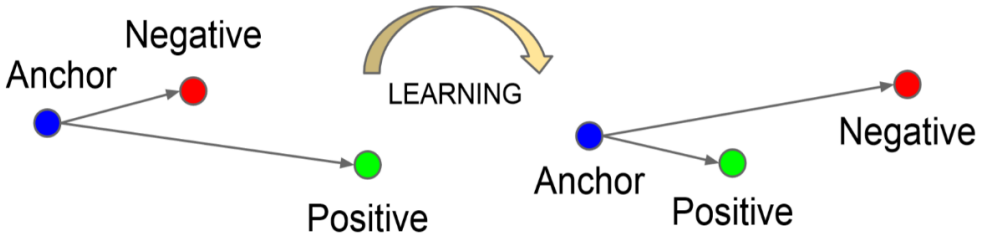
3.2.4 人脸识别

人脸识别核心算法采用 Facenet 进行识别，在 NCS 上加速运行，下面详细介绍人脸识别方案。

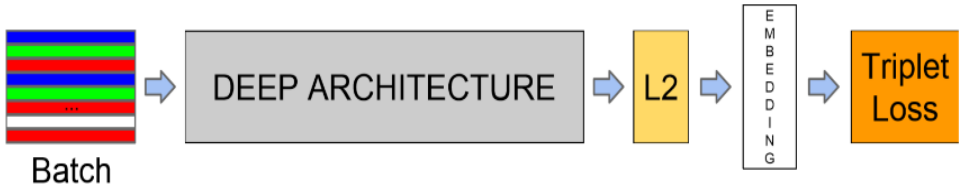
FaceNet

FaceNet[7]是针对人脸专门研究的网络，其可以直接将输入人脸图像通过深度卷积神经网络映射到欧几里得空间并得到高纬度的特征向量，根据空间距离大小判断不同图像间相似程度，最终达到人脸识别的目的。

相比于其他用于人脸识别的深度神经网络使用中间层作人脸图像向量映射，分类层作输出导致的低效、不直接，FaceNet 直接利用基于 triplets（三联子）-LMNN（最大边界近邻分类）的 loss 函数训练神经网络，网络直接输出 128 维特征向量，更高效。模型主要结构如图 3-15 所示。



Triplet Loss



模型总体结构

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

模型详细结构

图 3-15 FaceNet

采用 Adagrad 优化器、随机梯度下降法训练网络，FaceNet[1]在 LFW 数据集和 YouTube DB 数据集上测试，准确率分别为 99.63%和 95.12%。模型结构、层数、训练数据量、映射维度空间等对网络准确率都有较大影响，但模型对输入图像的质量并不敏感，较差质量下仍可以识别，即模型有较好鲁棒性，适合在真实场景中应用。

模型|NCS 准备

如前 3.2.1 所述，在 PC 机|服务器上搭建、训练出基于 tensorflow 的 facenet 标准模型并保存。然后通过 NCS-SDK 的 compiler 将标准模型转换为 NCS 的专用模型文件（.graph）并导入 PYNQ 中。

PYNQ 检查、开启 NCS 并将模型加载至 NCS，完成准备工作。如图 3-16 所示。

```
# 检查NCS
devices = mvnc.EnumerateDevices()
if len(devices) == 0:
    print('No NCS devices found')
    quit()
# 开启NCS
device = mvnc.Device(devices[0])
device.OpenDevice()
# 准备模型文件(加载至片上内存)——facenet.graph
graph_file_name = GRAPH_FILENAME
with open(graph_file_name, mode='rb') as f:
    graph_in_memory = f.read()
graph = device.AllocateGraph(graph_in_memory)
```

图 3-16 NCS 检测、开启

人脸录入

人脸录入可以通过提供目标照片也可以在摄像头前实时录入，本系统仅需提供目标单张照片即可达到较高检测率。获取目标图像后，构成一个图像列

表，将其输入神经网络，得到特征向量集合。每个目标对应一组特征向量（列表），所有目标特征向量构成一个字典，即人脸录入后的目标库。人脸录入部分程序如图 3-17 所示。

```
def realtime_input(frame, hdmi_out, graph, videoIn):

    ret,real_frame = videoIn.read()

    cv2.rectangle(frame, (0+offset, 0+offset),(frame.shape[1]-offset-1, frame.shape[0]-offset-1),rectangle_color, 10)
    cv2.putText(frame, 'Input...', (66, 88), cv2.FONT_HERSHEY_SIMPLEX, 1, text_color, 4)
    frame_out = hdmi_out.newframe()
    frame_out[:,:,:] = frame[:,:,:]
    hdmi_out.writeframe(frame_out)
    print('\r\n')
    print('!!!Please Input Name!!!')
    name = input()
    print('\r\n')
    cv2.imwrite('/home/xilinx/jupyter_notebooks/ncs-facenet_tensorflow/targets/' + name + '.jpg', real_frame)
    # 获取人脸目标路径
    temp_list = {}
    targets_list = os.listdir(targets_list_dir)
    targets_list = [i for i in targets_list if i.endswith('.jpg')]
    # ！！！人脸录入 ！！！（获取目标人脸特征集）
    targets_feature = facenet_ncs.feature(targets_list, temp_list, graph)
    cv2.putText(frame, 'Input...OK OK OK', (66, 88), cv2.FONT_HERSHEY_SIMPLEX, 1, text_color, 4)
    frame_out = hdmi_out.newframe()
    frame_out[:,:,:] = frame[:,:,:]
    hdmi_out.writeframe(frame_out)
    sleep(Delay)
    return targets_feature, targets_list

# ！（批量）特征提取 ！
def feature(targets_list, temp_list, graph):
    i = 0
    for target in targets_list :
        # 读取一个目标
        target = cv2.imread('/home/xilinx/jupyter_notebooks/ncs-facenet_tensorflow/targets/' + target)
        # 目标特征集合（字典）
        temp_list[i] = run_inference(target, graph)
        i = i + 1
    # 返回特征集合
    return temp_list
```

图 3-17　人脸录入

实时识别

由摄像头实时输入的待检测画面在图像预处理后，由 PYNQ 传至 NCS 输入神经网络（facenet），提取其特征向量（列表），然后将其特征向量与目标特征向量集合进行对比。此特征为 128 维向量，对比即在欧式空间计算待检测与目标特征向量的欧氏距离，欧氏距离越小，则说明相似度越高，若欧氏距离小于差异度阈值，则说明识别到目标。合理设置差异度阈值，使识别速度、精度、检测率综合达到较理想效果。本系统识别和未识别分别用绿框+名字和红框表示。实时识别部分程序如图 3-18 所示。

```
# 实时循环识别人脸
i = 0
while True:
    # 读入一帧图像
    ret, frame = videoIn.read()
    i = i + 1
    # 每隔5帧处理1帧图像
    if(i > 2):
        i = 0
        # ！！！人脸识别 ！！！
        frame_res = facenet_ncs.run_images(targets_feature, targets_list, graph, frame)
        # HDMI输出
        frame_res = cv2.resize(frame_res, (640, 480)) # 输出结果大小调整

        frame_out = hdmi_out.newframe()
        frame_out[:,:,:] = frame_res[:,:,:]
        hdmi_out.writeframe(frame_out)
```

```
# ！！！人脸识别（顶层模块）！！！
def run_images(targets_feature, targets_list, graph, input_frame):

    rect_width = 10
    offset = int(rect_width/2)

    l = len(targets_feature)
    f = 0
    i = 0

    # ！待检测图像特征提取 ！
    input_feature = run_inference(input_frame, graph)

    # ！！！人脸识别 —— 遍历目标特征集合（字典），循环比对 ！！！
#    for target_feature in targets_feature :
    while(i < l):
        matching = False
        # 匹配
        if (face_match(targets_feature[i], input_feature)):
            matching = True
            text_color = (0, 255, 0)
            match_text = "MATCH"
            #print('PASS!  File ' + input_image_file + ' matches ' + validated_image_filename)
            break;
        # 不匹配
        else:
            f =f + 1
        if f == l:
            matching = False
            match_text = "NOT A MATCH"
        i = i + 1

# 匹配 — 绿框+名字
if (matching):
    n = len(targets_list[i])
    # match, green rectangle
    cv2.rectangle(input_frame, (0+offset, 0+offset),(input_frame.shape[1]-offset-1, input_frame.shape[0]-offset-1),(0, 255, 0), 10)
    cv2.putText(input_frame, targets_list[i][:n - 4], (66, 88), cv2.FONT_HERSHEY_SIMPLEX, 1, text_color, 4)
# 不匹配 — 红框
else:
    # not a match, red rectangle
    cv2.rectangle(input_frame, (0+offset, 0+offset),(input_frame.shape[1]-offset-1, input_frame.shape[0]-offset-1),(0, 0, 255), 10)

return input_frame
```

图 3-18　实时识别

3.3 本章小结

本章从系统和模块层次上详细介绍了本课题人脸识别系统设计实现方案，总体包括图像输入、预处理、人脸录入、实时识别、输出显示等部分。该方案充分发挥并结合 PYNQ 和 NCS 的特性，总体逻辑为 PYNQ-ARM 负责调度和视频流输入、PYNQ-FPGA 负责图像预处理和视频流输出、NCS 负责神经网络推理识别。方案可实现在嵌入式终端低功耗下，较高速、高检测率的人脸识别。下一章将进行验证与分析，展示系统效果并进行结果分析。

4 验证与分析

4.1 系统搭建

如 3.1 系统设计所述，人脸识别系统总体包括包括图像输入（摄像头）、图像处理（主控）、图像输出（显示屏）等部分构成。系统如图 4-1 所示。

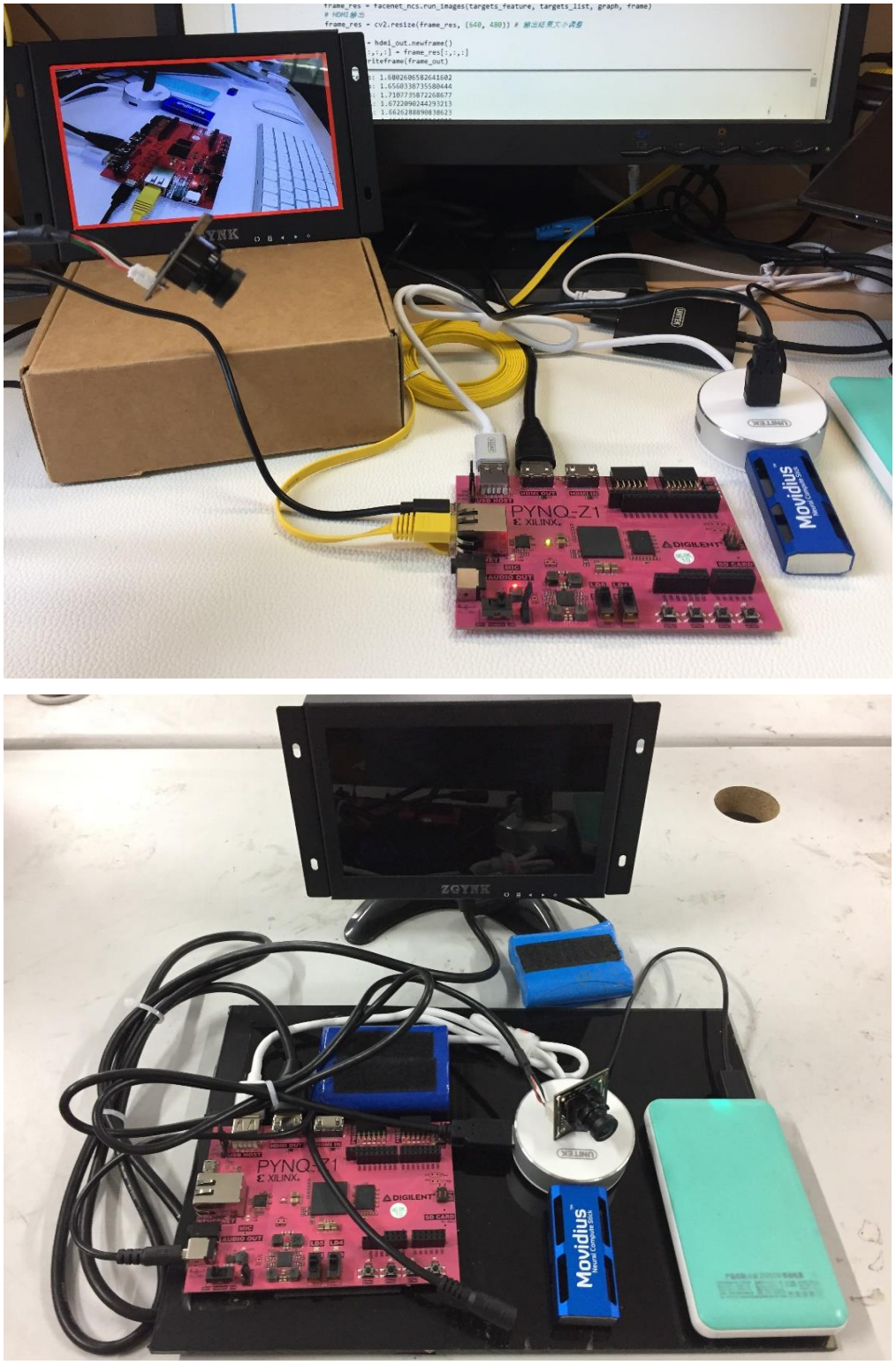


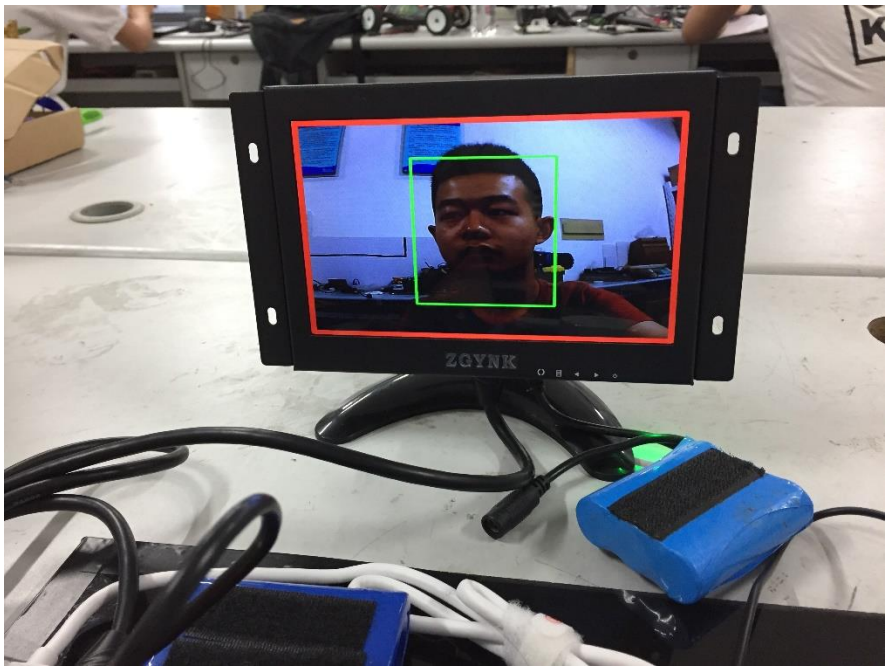
图 4-1　人脸识别系统搭建

4.2 系统效果|结果分析

由摄像头采集图像，人脸录入前无法识别，画面由红色框框出；人脸录入支持照片和实时录入，录入时画面由蓝色框框出；录入后再次由摄像头采集图像，人脸正常识别，画面由绿色框框处且显示目标名字。

系统整体运行流畅，检测较为准确，达到预期低功耗、实时、较高检测率识别人脸的目标。

人脸识别系统效果如图 4-2 所示。



人脸录入前

人脸录入

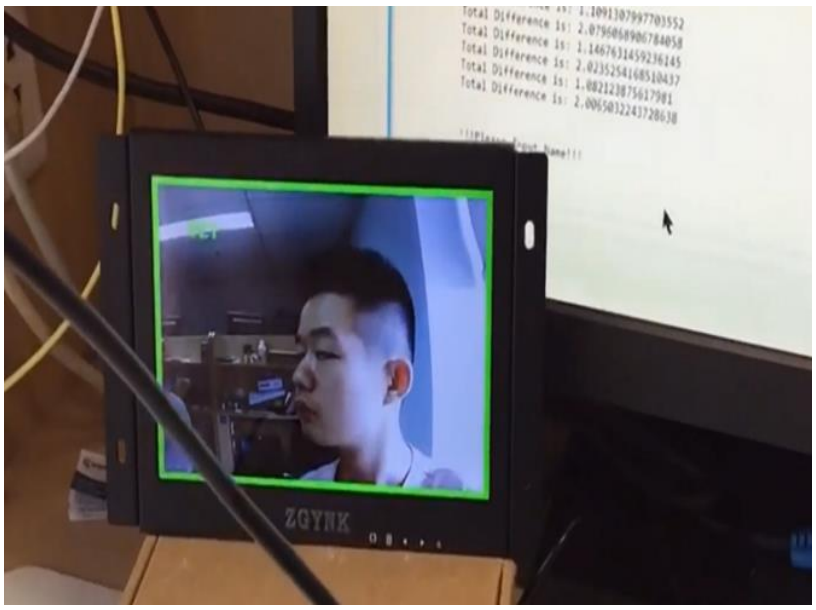
人脸录入后

图 4-2　人脸识别系统效果

系统采用在 PYNQ + NCS 上实现结合深度学习的人脸识别的方案[11,12]，在低功耗前提下，系统整体性能在实时性、检测率等方面取得较优性能。下面进行结果分析与对比。

1．检测率。人脸识别核心网络—facenet 在 LFW 数据集上取得 99.63% ± 0.09 的检测率；在 Youtube Faces DB 数据集上取得 95.12% ± 0.39 的检测率。在实际场景应用中，也取得较好的鲁棒性和泛化性，如图 4-3 所示，与传统人脸检测方法相比在较复杂环境背景、人脸有较大侧移|偏转等条件影响下，能达到更优检测率。

传统方案（特征+分类器）



本方案（PYNQ+NCS+DL）

图 4-3　人脸识别系统效果对比—检测率

2．检测速度。相比其他用于计算机视觉的深度神经网络，facenet 更加高效、直接，且在本课题所选嵌入式开发平台 PYNQ + NCS 的支持下，完全达到实时识别的目标要求。

如表 4-1 和图 4-4 所示，相比于目前一般在 ARM 上应用传统计算机视觉算法进行的人脸检测，本方案在检测率更优的前提下，实时性也取得更好效果。

一般方案在 ARM 上检测速度为 1.822fps,而本方案在 PYNQ + NCS 上检测速度最高达到 13.879fps,检测速度有明显的提升（PYNQ-ARM + NCS 方案检测速度为 4.454fps,PYNQ-FPGA + NCS 方案相对其有 3 倍以上的检测速度提升）

表 4-1 人脸识别系统效果对比—检测速度|实时性

平台	fps
ARM（一般方案）	1.822
PYNQ-ARM + NCS	4.454
PYNQ-FPGA + NCS	13.879

```
n_frames = 200

start_time = time.time()

for _ in range(n_frames):

    ret, frame_vga = videoIn.read()

    gray = cv2.cvtColor(frame_vga, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame_vga,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame_vga[y:y+h, x:x+w]

        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    # Output OpenCV results via HDMI
    frame_out = hdmi_out.newframe()
    frame_out[0:480,0:640,:] = frame_vga[0:480,0:640,:]
    hdmi_out.writeframe(frame_out)

end_time = time.time()

print('Runtime:',end_time - start_time,'FPS:',n_frames / (end_time-start_time))
```

```
Runtime: 109.74704885482788 FPS: 1.8223724654733784
```

ARM（一般方案）

```
n_frames = 200

start_time = time.time()

for _ in range(n_frames):

    ret, frame = videoIn.read()

    frame_res = facenet_ncs.run_images(targets_feature, targets_list, graph, frame)

    frame_res = cv2.resize(frame_res, (640, 480)) # 输出结果大小调整
    frame_out = hdmi_out.newframe()
    frame_out[:,:,:] = frame_res[:,:,:]
    hdmi_out.writeframe(frame_out)

end_time = time.time()

print('Runtime:',end_time - start_time,'FPS:',n_frames / (end_time-start_time))
```

```
Runtime: 44.901185512542725 FPS: 4.454225377727095
```

PYNQ-ARM + NCS

```
n_frames = 200

start_time = time.time()

for _ in range(n_frames):

    ret, frame = videoIn.read()

    frame_res = facenet_ncs.run_images(targets_feature, targets_list, graph, frame)

    frame_res = cv2.resize(frame_res, (640, 480)) # 输出结果大小调整
    frame_out = hdmi_out.newframe()
    frame_out[:,:,:] = frame_res[:,:,:]
    hdmi_out.writeframe(frame_out)

end_time = time.time()

print('Runtime:',end_time - start_time,'FPS:',n_frames / (end_time-start_time))
```

```
Runtime: 14.410114288330078 FPS: 13.87914044248549
```

PYNQ-FPGA + NCS

图 4-4　人脸识别系统效果对比—检测速度|实时性

4.3 本章小结

本章验证了课题设计实现方案，进行了本课题人脸识别系统的搭建、测试，并对测试结果进行了分析、对比。总体而言，本人脸识别系统在实时性、检测率、功耗等方面综合表现较为理想。

5 总结与展望

5.1 总结

本文主要包括引言、人脸识别算法、设计与实现、验证与分析、总结与展望等章节。

在引言里介绍了课题研究内容—人脸识别的背景、意义以及本课题的目的。即结合深度学习的计算机视觉算法是未来的趋势，人脸识别应用广泛、前景广阔，本课题要在嵌入式终端实现结合深度学习的人脸识别，达到综合要求。

在人脸识别算法一章详细介绍了传统和结合深度学习的人脸识别算法，分析了各自的优势与不足。传统人脸识别算法理论基础完备，但在精度上遇到了瓶颈；结合深度学习的人脸识别检测率得到大幅度提升，但是需要大量数据和较高算力的支撑，且目前深度学习理论基础相对薄弱，更像一个“黑箱”。此外，作者还选取了典型人脸识别算法在 PC 机端进行了实现、验证与对比，更直观体现人脸识别在用传统算法和深度学习算法实现时的特性。

设计与实现一章，先总体介绍了本课题人脸识别系统的设计、实现方案，包括系统硬件结构、程序开发、功能等。然后详细介绍了各个模块：主控部分 PYNQ+NCS 开发平台的特性、开发方法及在目标识别中的应用；图像的输入、输出，图像预处理模块的算法与实现；核心人脸识别模块（含模型、人脸录入、实时识别等部分）的介绍与设计等。

验证与分析一章，根据前章设计实现方案搭建了本课题人脸识别系统，并进行了系统效果测试与结果分析。人脸录入前无法识别，显示屏端用红色框框出，人脸录入支持提供目标照片和实时录入，录入后可识别到目标人脸，显示屏端用绿色框框出并显示目标 ID。经过分析对比，本方案人脸识别系统在检测率和检测速度上相对于一般方案均有较大改善。

总体来说本课题人脸识别系统在实时性、检测率、功耗等方面综合表现达到目标要求。最后进行课题总结展望。

5.2 展望

未来，本课题在速度、准确率、成本等方面还可进一步优化、改进。

算法方面，可以尝试更加轻量、高效的模型如 YOLO Tiny 来进行人脸检测，同时也可尝试 BNN[5]，均可提高检测速率；也可尝试使用更加复杂的网络如 SSD、ResNet 等进行人脸检测和识别，提高检测识别准确率，当然这对算力有较高要求；同时在人脸检测上可以考虑加入活体检测，以防照片欺诈，提高识别安全性。

实现方面，本课题采用 PYNQ + NCS，PYNQ 的 ARM 为双核，目前仅使用单核，

之后尝试双核、多线程运行，预计系统速度提升 2 倍以上；若放宽成本要求，可尝试 Nvidia 的嵌入式 GPU 开发平台—Jetson，其为 ARM + GPU 架构，在总体性能会有较大提升；同时考虑到成本、功耗，尝试突破 FPGA 开发，在仅使用 FPGA 的情况下搭建、推理神经网络甚至是神经网络的训练。

总体来说，本课题研究方向为结合深度学习的计算机视觉算法的嵌入式实现，后续在本课题基础上将做更加深入的研究与实现。

参考文献

[1] Huaizu Jiang.Face Detection with the Faster R-CNN[J].arXiv.2016.

[2] Shaoqing Ren,Kaiming He,Ross Girshick,Jian Sun.Faster R-CNN:Towards Real-Time Object Detection with Region Proposal Networks[J].arXiv.2016.

[3] Yufeng Hao,Steven Quigley.The implementation of a Deep Recurrent Neural Network Language Model on a Xilinx FPGA[J].2017.

[4] Andrew G.Schmidt,Gabriel Weisz, and Matthew French.Evaluating Rapid Application Development with Python for Heterogeneous Processor-based FPGAs[J].arXiv.2016.

[5] Yaman Umuroglu,Nicholas J.Fraser,Giulio Gambardella.FINN: A Framework for Fast, Scalable Binarized Neural Network Inference[J].arXiv.2016.

[6] Erwei Wang. PYNQ Classification - Python on Zynq FPGA for Neural Networks[D].2017

[7] Florian Schroff,Dmitry Kalenichenko,James Philbin.FaceNet:A Unified Embedding for Face Recognition and Clustering[J].CVPR.2015：815—823.

[8] DIGILENT.PYNQ-Z1 官方入门指导手册[Z].2017.

[9] 陈建生.人脸识别从 PCA 到 CNN[Z].2015.

[10] 梁淑芬，刘银华，李立琛.基于 LBP 和深度学习的非限制条件下人脸识别算法[J]. 通信学报. 2014，35（6）：154—160

[11] 曾爽，应骏，王健，曾维军. 基于 ZYNQ 的人脸检测的快速实现[J]. 视频应用与工程. 2015，39（15）：144—147

[12] 霍芋霖，符意德.基于 Zynq 的人脸检测设计[J].计算机科学.2016,43（10）：322—326

[13] 林妙真. 基于深度学习的人脸识别研究[D]. 2013（学术刊物文献）

[14] 骆超 苏剑波. 一种基于低功耗嵌入式平台的人脸识别系统[P]. 2016.

[15] 房晔，周湛. 一种嵌入式实时人脸识别装置[P]. 2016.

[16] 石浩然. 基于嵌入式系统人脸识别的分析与研究[D]. 2015

[17] 高 放，黄樟钦.基于异构多核并行加速的嵌入式神经网络人脸识别方法[J].计算机科学. 2018，45（3）：288—293

致谢

首先感谢郑州大学，感谢郑州大学信息工程学院，感谢郑州大学信息工程学院机器人实验室，大学四年，我在这里过得很充实、很难忘！

感谢郑州大学邓计才老师和北京大学曹健老师的悉心指导，本课题得以顺利完成。

自大二以来，我进入信息工程学院机器人实验室，在邓老师指导下开始参加机器人比赛、挑战杯、创新项目等科研竞赛。一路走来，有过困难、遗憾，但更多的是难忘、感恩与成长。我在这里收获了太多太多，老师的信任、比赛的经历、科研创新能力的提高、团队的管理经验等等。这些，是我的宝藏；这些，我十分感激！

自毕业设计工作启动以来，邓计才老师和曹健老师尽职尽责，严格把握毕设进度。邓老师每周都会召开毕设会议，检查各成员课题进度、督促进展，且要求每位学生进行周总结。曹老师为我提供实验环境和开发平台，并会定期来实验室检查我的课题进展情况，鼓励进步并及时指出其中的不足，经过和曹老师的讨论会使问题更加明晰、目标更加明确。

毕设期间，我遇到了很多困难，从最开始的算法、开发平台都不熟悉，到后来慢慢地入门并熟练应用，我体会到了学以致用的快乐与充实。还记得当时被一个难题卡住，纠结了好几天，仍然没有什么进展，甚至一度想申请更换题目。后来在开组会时，曹老师偶然提到一点，我听后顿时感觉重拾了希望。就是那一点思路的转换，之后毕设顺利进行。

感谢家人一直以来的支持，这是我的后盾且是我前进的源动力！

最后，再次感谢郑州大学，感谢信工，感谢老师、同学，感谢家人，感谢自己！

“我要尝遍世界这个园子里每棵树结的果实，我要心怀这份激情，走出校门，踏进世界。”我正是在寻找这份激情的路上，自强不息，厚德载物。

优秀是一种习惯，坚持总会有好结果！