硕士学位论文

面向边缘设备的高精度毫秒级人脸检测技术研究

RESEARCH ON HIGH-PRECISION MILLISECOND-LEVEL FACE DETECTION TECHNOLOGY FOR EDGE DEVICES

研　　究　　生：吴伟
指　导　教　师：于仕琪副教授

南方科技大学

二〇二三年六月

国内图书分类号：TP391.41　　学校代码：14325
国际图书分类号：004.8　　密级：公开

工学硕士学位论文

面向边缘设备的高精度毫秒级人脸检测技术研究

学位申请人：吴伟
指导教师：于仕琪副教授
学科名称：电子科学与技术
答辩日期：2023 年 5 月
培养单位：计算机科学与工程系
学位授予单位：南方科技大学

Classified Index: TP391.41
U.D.C: 004.8

Thesis for the degree of Master of Engineering

RESEARCH ON HIGH-PRECISION MILLISECOND-LEVEL FACE DETECTION TECHNOLOGY FOR EDGE DEVICES

Candidate:	Wu Wei
Supervisor:	Prof. Shiqi Yu
Speciality:	Electronics Science and Technology
Date of Defence:	May, 2023
Affiliation:	Department of Computer Science and Engineering
Degree-Conferring-Institution:	Southern University of Science and Technology

学位论文公开评阅人和答辩委员会名单

公开评阅人名单

无（全隐名评阅）

答辩委员会名单

主席	徐勇	教授	哈尔滨工业大学（深圳）
委员	张建国	教授	南方科技大学
	于仕琪	副教授	南方科技大学
	郑锋	副教授	南方科技大学
	王菡子	教授	厦门大学
秘书	罗胜达	博士后	南方科技大学

南方科技大学学位论文原创性声明和使用授权说明

南方科技大学学位论文原创性声明

本人郑重声明：所提交的学位论文是本人在导师指导下独立进行研究工作所取得的成果。除了特别加以标注和致谢的内容外，论文中不包含他人已发表或撰写过的研究成果。对本人的研究做出重要贡献的个人和集体，均已在文中作了明确的说明。本声明的法律结果由本人承担。

作者签名： 日期： 2023-6-6

南方科技大学学位论文使用授权书

本人完全了解南方科技大学有关收集、保留、使用学位论文的规定，即：

1. 按学校规定提交学位论文的电子版本。

2. 学校有权保留并向国家有关部门或机构送交学位论文的电子版，允许论文被查阅。

3. 在以教学与科研服务为目的前提下，学校可以将学位论文的全部或部分内容存储在有关数据库提供检索，并可采用数字化、云存储或其他存储手段保存本学位论文。

（1）在本论文提交当年，同意在校园网内提供查询及前十六页浏览服务。

（2）在本论文提交 ☑当年/ □ 一年以后，同意向全社会公开论文全文的在线浏览和下载。

4. 保密的学位论文在解密后适用本授权书。

作者签名： 日期： 2023-6-6

指导教师签名： 日期： 2023-6-6

摘　要

随着物联网技术的不断发展和普及，人脸检测作为一种重要的计算机视觉任务，在物联网应用中发挥着越来越重要的作用，是实现人与边缘设备智能交互的关键技术之一。然而，边缘设备上的人脸检测面临着诸多挑战，不仅需要适应低功耗、低性能、低存储等硬件条件的限制，还需要在保证高精度、高速度、高鲁棒性等性能指标的同时，适应各种复杂多变的场景和需求。为了解决这一难题，本文提出了一种超轻量级人脸检测算法 YuNet。本文提出轻量化模型设计原则，手动设计了主干网络、颈部网络和检测头，极大地降低了模型参数量和计算量，在保证模型效率和高准确率的同时，也提高了模型的简洁性和可控性。YuNet 还创新地采用了无锚框机制、自适应标签匹配策略和样本均衡的数据增强策略三个技术点，使得模型在毫秒级的检测速度下仍能保持高精度，并有效地解决了小目标检测、目标分布不平衡等问题。YuNet 不仅全面优于传统的基于手工特征的人脸检测算法，而且在同精度水平下与其他基于深度学习的轻量级人脸检测器相比，在参数量比较上具有数量级上的优势，并且在速度方面也有明显优势。据已知知识范围内，本文提出的 YuNet 几乎是参数量最少却仍保持高精度且速度最快的人脸检测器之一。本文还基于 YuNet 开发了 libfacedetection 开源项目，使用 C++ 实现且不依赖任何第三方库，在多个指令集平台 (AVX2/AVX512/NEON) 上进行了代码层面的 SIMD 指令优化加速，并实现了快速部署。实验结果表明，YuNet 在边缘设备上具有显著优势，并能够支持多种应用场景。

关键词：人脸检测；轻量级；边缘设备；计算机视觉

Abstract

With the development of Internet of Things technology, face detection plays an increasingly important role in IoT applications and is a key technology for human-edge device interaction. However, face detection on edge devices faces hardware constraints such as low power consumption, low performance, and low storage, while also requiring performance indicators such as high accuracy, high speed, and high robustness. To solve this problem, this paper proposes YuNet, a super-lightweight face detection algorithm. Based on the lightweight model design principles proposed in this paper, we manually design YuNet's backbone network, neck network and detection head to greatly reduce model parameter size and computation. For models with extremely small parameter size and computation, we innovatively propose anchor-free mechanism and adaptive label matching strategy as well as sample-balanced data augmentation strategy to achieve high accuracy while maintaining millisecond-level detection speed. YuNet not only outperforms traditional face detection algorithms based on handcrafted features but also has an order of magnitude advantage in parameter size over other lightweight face detection networks with the same accuracy based on deep learning. To our knowledge, our YuNet is almost the smallest parameter size yet still maintains high accuracy face detector. In this paper we also develop libfacedetection open source project based on YuNet which is implemented in C++ without relying on any third-party libraries performs code-level SIMD instruction optimization acceleration on multiple instruction set platforms and achieves fast deployment. Experimental results show that YuNet has significant advantages on edge devices and can support various application scenarios.

Keywords: face detection; lightweight; edge device; computer vision

目 录

第 1 章　绪　论

1.1　研究背景和意义

物联网 (Internet of Things, IoT) 是指把各种电子设备、机器设备、智能设备等连接到因特网上，并通过网络技术实现远程监测、管理和控制。物联网的意义在于，它把普通的物理世界与数字世界连接起来，使物理世界的信息和数据可以在网络上进行交互和共享。这样，我们就可以在网络上远程监测和控制物理世界的设备，更加方便、快捷地管理和控制它们。物联网为我们带来了诸多新的应用，如智能家居、智能医疗、智慧城市等，这些应用不仅提高了生活品质，而且帮助我们更有效地解决了很多问题。

人脸检测是计算机视觉领域的一个基础问题，旨在从图像或视频中自动识别和定位人脸，在物联网应用中具有重要的意义。在物联网环境下，人脸检测技术可以有效实现身份验证和访客管理。例如，在智能家居环境中，智能家居系统可以通过识别家庭成员的面部特征来自动识别每个人，以确定他们是否有权使用智能设备；可以通过识别家庭成员的面部特征来为每个人设置个性化的设置，例如调整温度、音量等；可以用来构建智能门铃系统，通过识别访问者的面部特征，以自动识别访问者，并在需要时发送警报，等等。

人脸检测技术通常需要处理大量的图像数据。如果采用云端计算模式，即将数据上传到云端服务器进行处理，会增加网络传输的延迟和成本，影响检测效果和用户体验，尤其是在网络不稳定或带宽不足的情况下。其次，云端处理人脸数据也存在一些安全和隐私的问题，如人脸数据的泄露、伪造、盗用等，可能导致个人信息被滥用或侵犯。因此，应当采用边缘计算模式来部署人脸检测，即在离用户最近的边缘设备上进行数据处理和分析，减少对云端资源的依赖。但是边缘设备的能耗以及存储和计算性能也限制了人脸检测技术的发展。边缘设备通常需要在低功耗、低成本、低延迟的条件下运行，而人脸检测算法和模型往往需要大量的数据和计算资源，这导致了两者之间的矛盾。所以，需要优化算法和模型，降低复杂度和开销，这样可以提高边缘设备的性能和效率，满足物联网应用中对实时性和准确性的要求，同时保护用户的隐私和安全。具体来说，优化人脸检测技术在边缘设备上部署的关键在于：设计极致轻量化、高效率、高精度的人脸检测算法，减少参数数量和计算量，适应边缘设备的硬件环境。

到目前为止，人脸检测技术的发展经历两个阶段，基于手工特征的传统方法

和基于深度学习的方法。这两种方法在参数量、计算量和准确率方面有着显著的差异。传统方法利用高效的算法来提取手工设计的特征，模型简单，占用存储空间小，运行速度快，适合在边缘设备上部署。但是传统方法需要大量的人工干预，训练数据量少，泛化能力弱，准确率低，难以应对复杂的场景变化。深度学习方法通过大规模的数据来自动学习特征，模型复杂，占用存储空间大，运行速度慢，依赖于特定的框架，不易在边缘设备上部署。但是深度学习方法具有很强的泛化能力，准确率高，在遮挡、光照变化等复杂场景下表现优异。两种方法都有广泛的实际应用场景。对于边缘设备，受限于计算能力、存储资源和功耗，通常只能采用传统方法来进行人脸检测，不得不在准确率上进行妥协，限制了很多深入应用的正常开展。因此，如何设计一个相对于传统方法，参数量更小、计算量更低、速度更快、准确率更高的基于深度学习的人脸检测算法，是工业界迫切需要解决的问题，也是具有重要实际意义和价值的研究课题。

1.2　国内外研究现状和分析

随着计算能力的飞速提升和创新性的传感、分析、渲染设备和技术的广泛应用，计算机越来越具有智能性。已有许多研究项目和商业产品展示了计算机能够通过摄像头和麦克风与人类进行自然交互，它们能够理解人类的输入并作出友好的反馈。实现这种人机交互的关键技术之一是人脸检测。人脸检测是所有基于人脸特征的分析算法的前提，例如人脸对齐、人脸建模、人脸重建、人脸识别、人脸验证、头部姿态跟踪、表情识别、性别/年龄识别等。给定一幅数字图像，人脸检测的主要目标是判断图像中是否存在人脸，并且如果存在，则给出人脸的位置和范围[1]。这对于人类来说可能是一件轻而易举的事情，但是对于计算机来说却充满了挑战性，这也使得人脸检测成为过去二十年里计算机视觉领域最热门的研究课题之一。影响人脸检测难度的因素包括尺度、位置、姿态、遮挡、光照、模糊等，在面对这些挑战时，过去二十年间研究者们不断地改进和优化了各种人脸检测算法，并取得了巨大的进步。

1.2.1　基于手工特征的传统人脸检测的研究现状

传统的人脸检测算法主要依赖于手工设计的特征。Yang[1]将这些早期方法分为四大类：基于知识的方法、特征不变方法、模板匹配方法和基于外观的方法。基于知识的方法[2]利用人类对典型面孔结构的先验知识来检测人脸。一张典型的人脸图像通常由两只相互对称的眼睛、一个鼻子和一张嘴组成，各部分特征之间的关系可以用它们的相对距离和位置来表示。该方法首先提取输入图像中的面部特

征，然后根据编码规则识别候选人脸，并通过验证过程减少错误检测。特征不变方法[3-6]旨在寻找对姿态和光照变化具有鲁棒性的人脸结构特征，该方法首先使用边缘检测器提取面部特征，然后推断出整张脸的存在，并根据提取的特征建立统计模型来描述它们之间的关系，并验证人脸是否存在。这些基于特征的算法存在一个问题，由于光照、噪声和遮挡，图像特征可能会严重损坏。面部边界可能会被削弱，而阴影可能会导致大量明显的错误边缘，这些都会使得感知分组算法失效。模板匹配方法[7-8]使用预先存储的人脸模板来确定图像中人脸的位置。给定一个输入图像，计算面部轮廓、眼睛、鼻子和嘴巴与标准模式之间相关值，并根据相关值判断是否存在人脸。该方法易于实现但不能有效地处理尺寸、姿势和形状变化等问题。为了实现尺度和形状不变性，后续提出了多分辨率、多尺度、子模板和可变形模板等技术。基于外观的方法[9-12]从一组代表性训练人脸图像中学习人脸模型，并依靠统计分析和机器学习技术来找到区分人脸和非人脸图像相关特征。所学到的特征以分布模型或判别函数形式呈现，并用于检测是否存在人脸。同时，考虑到计算效率和检测效果两方面因素，通常会对人脸图像进行降维处理。

2001 年，Viola 和 Jones[13]提出了一种开创性的人脸检测算法，使得人脸检测在实际场景中成为可能。该算法主要利用 Haar[14]特征和 Adaboost[15]算法来训练一个强分类器，用于检测图像中是否存在人脸。其检测过程主要包括四个步骤：首先，使用 Haar 特征来描述图像的局部区域，这些特征由黑白矩形块组成，反映了图像区域内的亮度差异。例如，一个 2×2 的 Haar 特征由两个黑色矩形和两个白色矩形构成，可以用来检测眼睛所在的位置。其次，使用 Adaboost 算法来训练一个强分类器，该分类器由多个弱分类器组合而成，每个弱分类器都是一个简单的二元分类器，用于判断图像区域是否包含人脸。在训练过程中，Adaboost 算法会对错误分类的样本进行加权处理，使得这些样本更容易被正确分类，在下一轮训练中得到更多关注，从而提高分类器的准确率。第三步，在检测新图像时，使用滑动窗口的方法对图像进行逐像素扫描，并在每个窗口内应用强分类器来判断该窗口是否包含人脸。通过调整窗口大小和位置可以检测不同尺度和方向上的人脸。最后一步，在输出结果时直接给出包含人脸区域的位置和大小信息。Viola-Jones 人脸检测器具有较高速度和准确性等优点，在实时应用中能够快速有效地检测出人脸。但是这种方法也存在一些问题：当图像中存在遮挡、光照不均匀、角度变化较大或表情复杂等情况时容易导致误检或漏检。

DPM[16-17]算法基于可变部件模型 (Deformable Part Model, DPM)，将目标对象看作由多个部件构成，并且每个部件都有自己独立且可变形的形状和特征表示方式。通过将这些部件组合起来构建目标对象整体的形状和特征表示方式，并利用

它们之间相互约束的关系进行目标对象识别与定位，从而提高了检测器对目标对象复杂变化情况下鲁棒性与准确性等方面表现能力。DPM 算法主要分为两个阶段：第一个阶段是生成候选框，根据输入图像及其尺度空间金字塔等信息生成一系列可能包含目标对象的候选框。第二个阶段是训练和检测，对于人脸的每个部件 (如眼睛、鼻子、嘴巴等)，学习其形状和特征表示方式，并用支持向量机 (Support Vector Machines, SVM) 训练出相应的部件分类器。然后，对于每个候选框，通过组合各个部件分类器得到整体人脸分类器，并计算其得分，用于判断该候选框是否包含人脸。在检测过程中，通常采用滑动窗口的方式进行。对于检测到的每个人脸区域，通过比较其得分，筛选出得分较高的区域作为最终结果。与 Viola-Jones 人脸检测器相比，DPM 算法能够更好地处理遮挡、光照不均匀、角度变化等复杂情况下的人脸检测问题，并且对尺度变化范围大的图像有更强的适应能力。当然，DPM 算法也存在一些缺点：计算复杂度高、检测速度慢等。

这些传统方法都有一个共同问题：它们都依赖于手工设计的特征来描述图像中的人脸信息，这些特征可能不足以捕捉复杂多样化的人脸变化情况，并且难以适应不同场景和数据集下的需求变化，因此限制了它们在实际应用中表现出来的性能。此外，这些方法通常需要调整大量参数来优化模型效果，这需要花费大量时间和经验，并且难以充分利用大规模人脸数据集所提供优势。尽管如此，在一些资源受限或实时性要求高等场景下，传统方法仍然具有一定价值，因为它们通常具有较高效率、较低计算成本和相对可接受精度等特点。例如，在大多数数码相机中都内置了基于 Viola-Jones 算法实现的人脸检测功能，用于自动对焦和拍照。因此，传统方法并不是完全过时或无用，而是在一定程度上被深度学习方法所补充和超越。

1.2.2 基于深度学习的人脸检测研究现状

随着深度学习技术的发展，基于深度学习的人脸检测方法能够自动学习图像中的复杂特征，相比于传统方法在检测准确率上取得了显著优势，逐渐成为研究的主流。Feng[18] 将基于深度学习的人脸检测器架构分为三类，多阶段检测架构、两阶段检测架构和单阶段检测架构。

受传统人脸检测方法的影响，早期深度学习的人脸检测大都采用级联结构。Girshick 等[19] 在 2015 年提出了 Faster-RCNN，以突破性的速度和精度性能成为目标检测领域的里程碑。Faster-RCNN 引入了一个称为区域建议网络 (Region Proposal Network, RPN) 的子网络，它可以在图像中生成候选区域，这些区域用于检测目标。RPN 使用卷积神经网络 (Convolutional Neural Networks, CNN) 来提取图像特征，并根据这些特征生成候选区域。RPN 输出每个候选区域的得分和边界框，这些得分

可以用来过滤掉低质量的候选区域，并将其提供给后续的分类器和回归器。整个模型采用端到端的训练方式，可以同时学习特征提取、候选区域生成、目标分类和边界框回归。之后，一些研究人员基于人脸数据改进了 Faster-RCNN。同年，Li 等[20]提出了 CascadeCNN。其主要方法是采用级联的 CNN 来进行人脸检测。该模型包括三个级联的 CNN 分类器，每个分类器都使用了级联的检测器。在级联的过程中，前一个分类器用于过滤掉大量的负样本，以提高后续分类器的效率和准确率。Zhang 等[21]在 2016 年提出了 MTCNN，在单个模型中完成了人脸检测和人脸关键点定位两项任务，在多个公开基准数据集中取得了非常好的效果。它由三个级联的卷积神经网络实现，Proposal Network (P-Net) 生成候选框，Refine Network(R-Net) 和 Output Network(O-Net) 对候选框进行进一步地筛选和精细化调整。Wang 等[22]在 2017 年提出了基于 Faster-RCNN 架构的 Face-RCNN。其主要贡献在于将人脸识别中使用的 Center Loss 引入到人脸检测中，并提出在线困难样本挖掘算法 (Online Hard Example Mining, OHEM)。另外，为了弱化人脸尺度变化范围大的影响，Face-RCNN 采用多尺度训练策略，即在训练阶段将输入图片进行多尺度缩放。无论是级联的多阶段检测网络架构还是两阶段检测网络架构，其计算速度在很大程度上依赖于图像中人脸的数量，人脸数量的增加也将增加网络内部传递到下一阶段的候选区域数量，导致整体推理时延较高且不稳定。

由于一些实际应用场景需要人脸检测任务实时进行，单阶段的人脸检测器更受欢迎，因为它们的处理时间不受图像中人脸数量的影响。单阶段架构只使用一个神经网络来直接生成预测框，通常包括两个部分：特征提取和预测框生成。特征提取部分使用卷积神经网络从原始图像中提取特征，预测框生成部分则将这些特征输入到几个并行的卷积层中，以生成预测框。它们不需要生成候选区域，而是使用密集锚点设计来替代多阶段检测网络架构中的候选区域建议。然而，单阶段人脸检测器也面临着一些挑战：如何处理尺度变化范围大、姿态、光照和遮挡极端变化的人脸；如何平衡正负样本比例，避免过拟合或欠拟合；如何设计有效的特征增强模块和优化方法，提高模型性能。针对这些挑战，一些研究人员提出了各种改进方案。Hu 等[23]提出了 HR 方法，它是最早的基于锚框机制的单阶段人脸检测器之一，它根据不同尺度范围内的人脸分别训练不同的检测器，并从人脸尺度、分辨率和特征上下文信息三个方面进行优化。Najibi 等[24]提出了 SSH 方法，它直接在不同层次的特征映射上构建具有丰富感受野的检测模块。Zhang 等[25]提出了 S3FD 方法，它引入了一种锚框补偿策略，并使用额外的子网络来增强多尺度下的人脸检测结果。Tang 等[26]提出了 PyramidBox 方法，它采用了一种“数据-锚框”的采样策略来增加小人脸在训练数据中的比例，并使用一个称为 LFPN 的子

网络来利用低层次特征进行小人脸检测。Li 等[27] 提出了 DSFD 方法，在主干网络上引入了小人脸监督信号，并使用一个称为 EFPN 的子网络来显著提高特征金字塔性能。Deng 等[28] 提出了 RetinaFace 方法，在原有目标类别和边界框回归任务之外增加了额外任务：关键点回归任务，并采用可变形卷积模块来增强上下文信息。Liu 等[29] 发现训练阶段未匹配到真实目标区域但具有较强定位能力的锚框也可以作为输出候选区域，并基于此设计了一种在线高质量锚框挖掘策略；同时在另一项工作[30] 中采用单路径搜索算法对主干网络和颈部网络进行联合优化。Li 等[31] 在 ASFD 方法中提出了一种用于优化特征增强模块的差分架构搜索算法，从而实现高效的多尺度特征融合和上下文增强。这些方法都对人脸检测任务精度和速度方面进行了深入探索，但大多数都是通过使用复杂而沉重的特征增强模块、稠密而技巧性的锚框设计以及复杂而耗时的训练测试策略实现极高精度排名优势。然而，在真实场景中应用时，高昂运行成本和较长推理时延仍然是障碍。

为了提高人脸检测模型的实际应用效果，研究人员提出了许多轻量化的方法。例如，Zhang 等[32] 提出了 FaceBoxes 方法，它是最早实现实时检测的单阶段人脸检测方法之一。它的贡献在于提出快速消化 (RDCL) 模块和多尺度卷积 (MSCL) 模块来实现快速而有效的特征提取。He 等[33] 提出了 LFFD 方法，它引入残差结构进行特征提取，并根据感受野和有效感受野来设计基于锚框机制的检测策略。YOLO5Face[34] 基于 YOLOv5[35] 实现专用的轻量级人脸检测器。Guo 等[36] 提出了 SCRFD 方法，它引入样本再分配策略和计算再分配策略来增强训练样本丰富性和优化计算量分配，并且根据不同的计算量标准 (0.5/3.0/10.0 GFLOPs 等) 提出一系列高效人脸检测器。这些方法都大大降低了人脸检测任务的推理延迟，但是也牺牲了一定的检测精度，并且还有进一步轻量化的空间。

1.3　论文的主要研究内容和组织结构

本学位论文以边缘设备为应用场景，设计并实现了一种轻量化毫秒级人脸检测算法，并在开源项目中进行了部署。全文共分为五个章节，具体内容如下：

第一章：绪论。阐述了本研究的背景、意义、挑战和难点，并回顾了基于手工特征和基于深度学习的人脸检测方法的发展现状和存在的问题。

第二章：人脸检测相关理论概述。介绍了深度神经网络和卷积神经网络的基本原理和发展历程，并重点分析了卷积神经网中常用的基本单元和损失函数。此外，还介绍了人脸检测任务所涉及的常用数据集。

第三章：YuNet 超轻量级人脸检测算法设计。首先梳理了深度学习人脸检测任务流程和主要结构，并总结了轻量化网络模型设计的规律。然后从主干网络、颈部

网络和检测头网络三个方面详细阐述了 YuNet 算法的网络结构设计。接着介绍了 YuNet 算法所采用的训练策略，包括无锚框机制、自适应标签匹配策略和多任务联合损失函数。最后通过与其他轻量级人脸检测算法进行对比实验，验证了 YuNet 算法在模型参数、计算量和推理时延等方面的优势。

第四章：基于 YuNet 算法在边缘设备上的部署实现。分别介绍了训练库和运行库两个开源项目。训练库主要说明了训练程序搭建过程及模型导出方法。运行库主要介绍了 SIMD 指令集优化卷积运算过程及其优缺点，并使用代码示例说明优化细节。最后在不同指令集平台和不同输入分辨率下进行对比实验并分析结果。

第五章结论与展望，总结本论文完成的主要工作与创新点，并指出不足之处与改进方向。

第 2 章　人脸检测相关理论概述

本论文的主要研究内容为基于卷积神经网络的超轻量级人脸检测算法，因此，本章节首先对卷积神经网络的基本单元和损失函数进行简要介绍。接着对人脸检测相关的数据集和评估标准进行梳理。最后对经典的人脸检测算法进行分析和总结。

2.1　深度神经网络介绍

深度神经网络 (Deep Neural Network，DNN) 是一种基于多层神经元构建的机器学习模型。它由多个神经元和多个层组成，每层的输出被作为下一层的输入。每个神经元通过一个非线性函数 (称为激活函数) 将其输入转换为输出。深度神经网络可以处理大规模的复杂数据，例如图像、音频、文本等，可以用于分类、回归、聚类、生成等各种任务。深度神经网络的深度是指其拥有的层数，深度越大，网络可以学习到更多的抽象特征，从而提高模型的性能。与传统的浅层神经网络相比，深度神经网络具有更高的准确性和更好的泛化能力，能够处理更加复杂的任务。深度神经网络的训练过程通常采用反向传播算法 (Back Propagation) 和随机梯度下降 (Stochastic Gradient Descent, SGD) 等优化算法来更新模型的参数。卷积神经网络 (Convolutional Neural Network，CNN) 是一种特殊的深度神经网络，可以自动学习和提取输入图像的特征，从而实现对图像的分类、检测、分割等任务。

2.1.1　卷积神经网络的发展

卷积神经网络的最早雏形可以追溯到上世纪 80 年代，但由于当时计算能力有限，加之数据集也不充足，所以卷积神经网络的发展受到了很大的限制。这个阶段最著名的模型是 LeNet-5[37]，它被广泛应用于手写数字识别。随着计算机硬件和大规模数据集的发展，深度卷积神经网络开始被广泛应用于图像分类、目标检测等任务。2012 年，AlexNet[38] 在 ImageNet[39] 比赛中斩获冠军，标志着卷积神经网络的进入深度化阶段。在这个阶段，一些重要的技术如批量归一化、残差连接、深度可分离卷积等也被提出，进一步改善了卷积神经网络的性能，同时一些经典的基础模型如 VGG[40]，ResNet[41] 等也相继提出，启发了大量的后续工作。在深度化的基础上，研究者们开始探索如何进一步优化卷积神经网络。其中一项重要的进展是“迁移学习”，即利用预训练的神经网络模型，通过微调的方式来训练新

的任务。随着卷积神经网络性能的不断提高，其应用范围也不断扩展。除了图像领域，卷积神经网络也被广泛应用于语音识别、自然语言处理、推荐系统等领域。由于卷积神经网络模型的计算量通常非常大，因此如何加速模型训练和推理一直是一个重要的研究方向。硬件加速技术如 GPU、TPU 等得到了广泛研究和应用，大大提升了模型的训练和推理速度。一些新型的硬件加速技术如神经处理器 (Neural Processor) 等也在研究中，有望进一步提升卷积神经网络的性能。

2.1.2　卷积神经网络基本单元

卷积神经网络的基本单元有卷积层 (Convolutional Layer)、池化层 (Pooling Layer)、激活层 (Activation Layer)、批量归一化 (Batch Normalization) 和全连接层 (Fully Connected Layer) 等。下面对本文将涉及的一些基本单元进行简要介绍。

2.1.2.1　标准卷积

卷积运算是卷积神经网络中最基本的计算单元，也被称为互相关 (Cross-correlation) 运算。最常用的卷积类型是二维标准卷积层，通常缩写为 Conv2D。它的原理是将一个二维卷积核在输入数据矩阵上滑动，每次选取与卷积核大小相同的子矩阵，然后对应位置相乘并求和得到一个输出像素。这样就得到了一个新的二维输出矩阵。为了保持输入和输出矩阵的尺寸一致，通常会在输入矩阵的边界上填充一些值，这个过程叫做边界填充 (Padding)。常用的填充值是 0。通过边界填充，可以让卷积核扫描到输入数据的边缘信息，从而避免信息损失。卷积核滑动时每次移动的行数和列数叫做步长 (Stride)。我们可以通过调整边界填充和步长的大小来控制输出矩阵的尺寸。

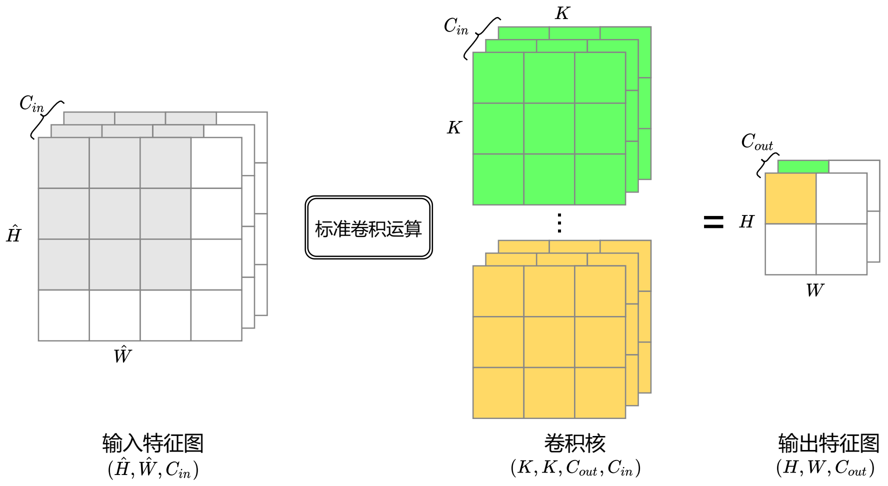


图 2-1　标准卷积计算过程

图 2-1展示了标准卷积的计算过程。假定输入特征图的尺寸为 $(\hat{H},\hat{W})$，输入通

道数为 C_{in}，那么对于每个输入通道，都有一个对应的 $K \times K$ 的卷积核进行逐点相乘并求和并得到一个值，单次的计算量为 $K \cdot K \cdot C_{in}$。然后，对整个特征图进行滑动，假设输出特征图的尺寸为 (H, W)，那么一个卷积核处理输入数据的计算量为 $K \cdot K \cdot H \cdot W \cdot C_{in}$。假设在某个标准卷积层使用 C_{out} 个卷积核，那么整个卷积层的计算量计算公式为：

$$\#FLOPs = K \cdot K \cdot H \cdot W \cdot C_{in} \cdot C_{out} \tag{2-1}$$

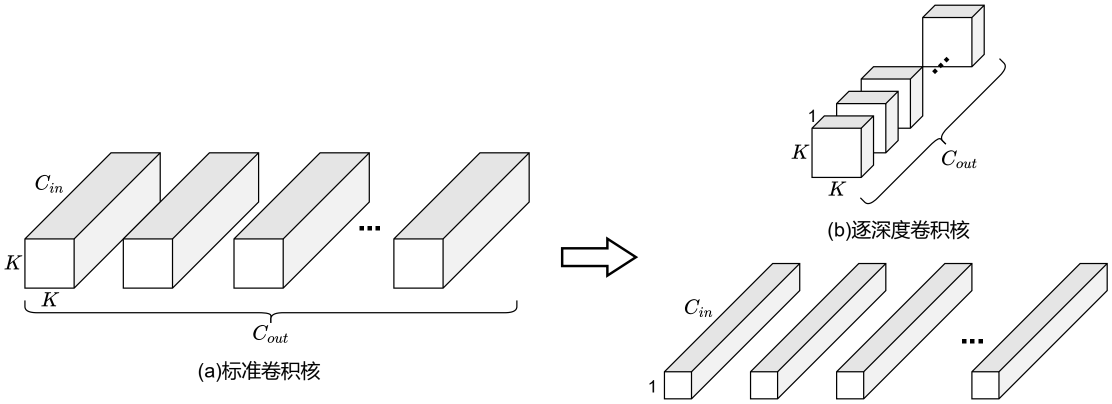
这种情况下卷积核的参数量计算公式为：

$$\#Params = K \cdot K \cdot C_{in} \cdot C_{out} \tag{2-2}$$

值得注意的是，卷积层的参数量和计算量与卷积层本身参数相关，后者还与输出特征图的尺寸相关，两者都与输入特征图尺寸无关。

2.1.2.2　深度可分离卷积

深度可分离卷积 (Depthwise Separable Convolution) 是一种将标准卷积分解为两个子卷积的方法，分别是逐点卷积 (Pointwise Convolution) 和逐深度卷积 (Depthwise Convolution)。这样可以减少模型的参数和计算量，同时保持良好的特征提取能力。深度可分离卷积在轻量级模型 Mobilenet[42-44] 和 Xception[45] 中有广泛的应用。



(a)标准卷积核

(b)逐深度卷积核

(c)逐点卷积核

图 2-2　标准卷积转化为深度可分离卷积的示意图

图2-2展示了标准卷积转化为深度可分离卷积的过程。逐点卷积就是卷积核大小为 1×1 标准卷积，它可以将输入特征图的每个通道进行加权组合，然后输出指定数量通道的特征图。它的作用是实现不同通道之间的信息交换和对齐。逐深度卷积就是将输入输出通道数设为相同的分组卷积，它的计算过程如图2-3所示，输入特征图为 (H, W, C)，使用 C 个 $K \times K$ 的卷积核分别对对应通道的特征图进行卷

积，得到通道数为 C 的输出特征图。它的作用是独立地提取每个通道上的感受野信息。

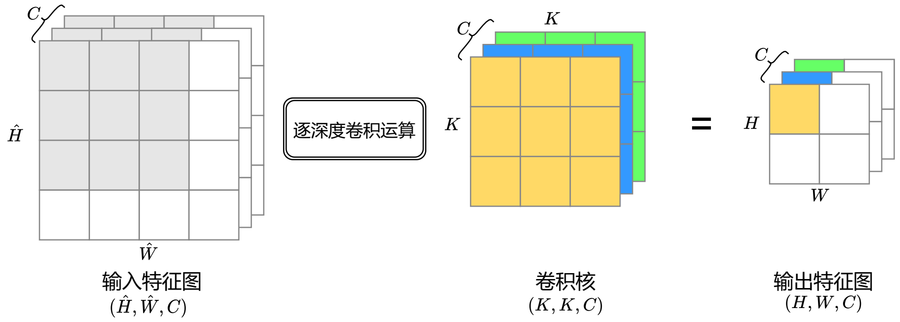


图 2-3　逐深度卷积计算过程

假设标准卷积核为 (K,K,C_{out},C_{in})，转化得到卷积核为 $(1,1,C_{out},C_{in})$ 的逐点卷积和 (K,K,C_{out},C_{out}) 的逐深度卷积。假设输出特征图尺寸为 (H,W)。忽略偏置对结果的影响。对于逐点卷积，由公式2-1取 $K=1$ 得到的计算量为：

$$\#FLOPs = H \cdot W \cdot C_{in} \cdot C_{out} \tag{2-3}$$

由公式2-2得到参数量为：

$$\#Params = C_{in} \cdot C_{out} \tag{2-4}$$

对于逐深度卷积，由于输入特征图的每个通道仅使用一个卷积核进行卷积输出，所以参数量公式为：

$$\#Params = K \cdot K \cdot C_{out} \tag{2-5}$$

相应的，计算量公式为：

$$\#FLOPs = K \cdot K \cdot H \cdot W \cdot C_{out} \tag{2-6}$$

深度可分离卷积的参数量和计算量分别为逐点卷积和逐深度卷积的参数量与计算量之和，那么总参数量为：

$$\begin{aligned} \#Params &= K \cdot K \cdot C_{out} + C_{in} \cdot C_{out} \\ &= (K^2 + C_{in}) \cdot C_{out} \end{aligned} \tag{2-7}$$

总计算量为：

$$\begin{aligned} \#FLOPs &= H \cdot W \cdot C_{in} \cdot C_{out} + K \cdot K \cdot H \cdot W \cdot C_{out} \\ &= (K^2 + C_{in}) \cdot C_{out} \cdot H \cdot W \end{aligned} \tag{2-8}$$

因此转化后的的深度可分离卷积和相应标准卷积的参数量之比为：

$$\frac{(K^2 + C_{in}) \cdot C_{out}}{K^2 \cdot C_{in} \cdot C_{out}} = \frac{1}{C_{in}} + \frac{1}{K^2} \tag{2-9}$$

计算量之比为：

$$\frac{(K^2+C_{in})\cdot C_{out}\cdot H\cdot W}{K^2\cdot C_{in}\cdot C_{out}\cdot H\cdot W}=\frac{1}{C_{in}}+\frac{1}{K^2} \tag{2-10}$$

可以看到对应的计算量和参数量的比值是相同的。本课题的所有卷积核大小 K 都为 3，而输出通道数 C_{out} 往往远大于 K^2，可以近似认为深度可分离卷积替换标准卷积的计算量和参数量的压缩比为 $K^2=9$。所以这个分解过程能极大的减少计算量和参数量。MobileNet[43] 论文中指出，在图像分类任务上替换所有的标准卷积为深度可分离卷积仅会造成轻微的精度下降，相对于计算量和参数量的数量级的减少无疑是可以接受的。由于现有的深度学习计算库对深度可分离卷积的加速优化较少，所以我们在实验中发现在 CPU 上计算速度提升远比在 GPU 上的提升大。因此对于部署到边缘设备的人脸检测模型，如果将标准卷积替换为深度可分离卷积，尽管其模型大小将会显著降低，但得到的模型精度可能是次优的。

2.1.2.3　池化层

池化层是卷积神经网络中常用的一种层，它一般紧接在卷积层之后。池化层的主要功能是对卷积层输出的特征图进行降采样，从而减少网络的参数个数，降低模型复杂度，同时还可以防止过拟合。池化层将输入特征图按照固定大小的窗口 (称为池化窗口) 从左到右，从上到下进行固定步长滑动。池化运算是预先确定的，在每个池化窗口内，池化层通常使用最大值池化或平均值池化的方式对窗口内的值进行聚合，得到的值作为该窗口的输出。图2-4是最大值池化运算的示意图，输入特征图尺寸是 4 × 4 的，池化窗口为 2，步长为 2。则对每个滑动的池化窗口取最大值，输出尺寸为 2 × 2 的特征图。对于多通道的输入特征图，池化层通常对每个通道进行单独处理，得到相应的多个输出特征图。在大多数情况下，最大值池化比平均值池化有更好的性能。另外，有些工作使用步长较大的卷积层或者其他方法 (如空洞卷积等) 来代替池化层进行降采样。但是，使用其他卷积来进行降采样会增加参数量和计算量。由于池化运算不涉及任何权重参数，计算复杂度也非常低，所以本文广泛使用最大值池化层来进行降采样。

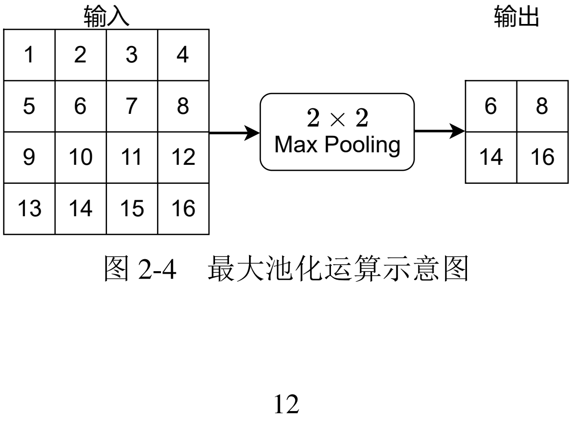


图 2-4　最大池化运算示意图

2.1.2.4　激活层

在 CNN 中，激活函数一般使用饱和非线性函数 (Saturating Nonlinearity) 如 sigmoid 函数、tanh 函数等。饱和非线性函数的特点是当输入超过一定范围时，输出就会趋于一个常数值，不再随输入变化。相比之下，不饱和非线性函数 (Non-saturating Nonlinearity) 能够避免梯度爆炸/梯度消失问题，同时也能够加快收敛速度[46]。Jarrett 等人[47]探讨了卷积神经网络中不同的修正非线性函数 (Rectified Nonlinearity)，包括 max(x, 0) 非线性函数，通过实验发现它们能显著提升卷积网络的性能，Nair 等[48]也验证了这一结论，因此目前的 CNN 结构中常用不饱和非线性函数作为卷积层的激活函数如 ReLU 函数，及其变体函数。ReLU 函数的计算公式如下:

$$f(x) = \max(0, x) \tag{2-11}$$

对于 ReLU 而言，如果输入不大于 0，则输出与输入相等，否则输出为 0。使用 ReLU 函数，输出不会随着输入的逐渐增加而趋于饱和。

2.1.2.5　批量归一化层

神经网络训练时，为了提高数据的稳定性和效率，常用的一种方法是标准化处理。它的作用是将输入数据的不同维度变换为均值为零，方差为一的正态分布。这样可以消除数据量纲的影响，使得各个特征的分布更加接近，并加速模型拟合过程。但是对于深度神经网络，由于模型参数在训练中不断更新，会导致每层输入数据的均值和方差发生变化。这种现象叫做内部协变量漂移 (Internal Convariate Shift)，它会降低网络收敛速度，并可能引起梯度消失 (Vanishing Gradient) 问题。针对这个问题，Ioffe 等[49]提出了批归一化 (Batch Normalization, BN) 层。BN 层在训练时对隐藏层特征做标准化处理，并通过两个线性参数进行缩放平移作为新输入。神经网络会迭代更新 BN 层参数。BN 层具有与卷积层相同权重共享特性，即特征图中同一通道像素共享一组 BN 参数。通过调整隐藏层神经元输入，BN 层能够保证网络训练稳定性，并加快收敛速度，在一定程度上缓解梯度消失或爆炸问题。虽然 BN 层在训练时有效果，在推理时却增加了运算量和内存或显存占用。因此我们需要将 BN 层参数合并到卷积层，提高模型推理速度。

假设 x 为网络的某一激活层特征，假模型训练批次中共有 n 个样例，那么其特征分别是 $x_1, x_2, ..., x_n$，则采用下列公式进行归一化：

$$\hat{x}_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2-12}$$

其中 μ 和 σ^2 是当前批次上计算得到的均值和方差，ϵ 防止除零所设置的一个极小

值，γ 是放缩系数，β 是平移系数。在训练过程中，μ 和 σ 在当前批次上计算：

$$\mu = \frac{1}{n}\sum x_i \tag{2-13}$$

$$\sigma^2 = \frac{1}{n}\sum (x_i - \mu)^2 \tag{2-14}$$

参数 γ 和 β 由和其它模型参数一起通过梯度下降方法训练得到。在测试阶段，我们不能对一个批次图像进行预测，一般是对单张图像测试。因此，通过前面公式计算 μ 和 σ 就不可能。BN 的解决方案是采用训练过程中指数移动平均值 $\hat{\mu}$ 和 $\hat{\sigma}^2$。

在卷积层中，训练得到的卷积权重为 W，偏置为 B，则 BN 层和卷积层的融合公式为:

$$W_{merge} = W\frac{\gamma}{\sqrt{\hat{\sigma}^2 + \epsilon}} \tag{2-15}$$

$$B_{merge} = (B - \hat{\mu})\frac{\gamma}{\sqrt{\hat{\sigma}^2 + \epsilon}} + \beta \tag{2-16}$$

2.1.3　损失函数

人脸检测的损失函数由分类损失和回归损失两部分构成，下面简要介绍本论文中用到的损失函数。

2.1.3.1　Smooth-L1 损失函数

Smooth-L1 Loss 是深度学习中用于回归任务的一种损失函数，最早在 Fast RCNN[50] 中提出，它可以用于替代传统的均方误差 (Mean Squared Error, MSE) 损失函数。相比于 MSE 损失函数，Smooth-L1 Loss 具有更好的鲁棒性。其定义为：

$$L_{\text{smooth}} = \begin{cases} 0.5(x_i - y_i)^2, & |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & |x_i - y_i| \geqslant 1 \end{cases} \tag{2-17}$$

其中 x_i 是模型预测的值，y_i 是真实值，i 表示第 i 个样本。如果预测值和真实值之间的差小于 1，则使用平方差的方式计算损失；如果差大于等于 1，则使用绝对值的方式计算损失。为了使损失函数曲线连续可导，在差大于等于 1 的时候减去 0.5，并在差小于 1 时乘以 0.5。相比于 MSE 损失，Smooth-L1 Loss 的优势在于对于异常值的处理能力更好，因为平方函数的梯度在接近 0 的时候会变得非常小，会导致在优化过程中，梯度几乎为 0，使得模型无法继续学习。而 Smooth-L1 Loss 在接近 0 的时候，梯度是线性的，因此不会出现这种情况。在本文中，Smooth-L1 Loss 用于人脸关键点的回归。

2.1.3.2　交并比损失函数

交并比 (Intersection over Union，IoU), 常用于描述两个图形的重叠程度，其计算公式如下：

$$IoU = \frac{area(A) \cap area(B)}{area(A) \cup area(B)} \tag{2-18}$$

其中 $area(x)$ 代表图形的面积，则图形之间重叠的区域越大，IoU 值越大。在人脸检测中，训练阶段使用预测人脸框和实际人脸框之间的交并比来区分正样本和负样本，测试阶段使用预测人脸框之间的交并比来决定是否进行非极大值抑制。对于预测的人脸框，由于框的四个值 (通常为左上角点横坐标值、左上角点纵坐标值、框宽度、框长度，即 (tx, ty, w, h) 需要看作一个整体回归，Yu 等[51]使用 IoU 损失来替代 Smooth-L1 损失来计算目标框的回归。假设 $(x_1^t, y_1^t, x_2^t, y_2^t)$ 表示目标框，$(x_1^p, y_1^p, x_2^p, y_2^p)$ 表示预测框，则目标框和预测框的面积分别为：

$$\begin{gathered} S^t = (x_2^t - x_1^t)(y_2^t - y_1^t) \\ S^p = (x_2^p - x_1^p)(y_2^p - y_1^p) \end{gathered} \tag{2-19}$$

假设 (x_1, y_1, x_2, y_2) 表示重叠区域，则重叠区域面积 I_{std} 为：

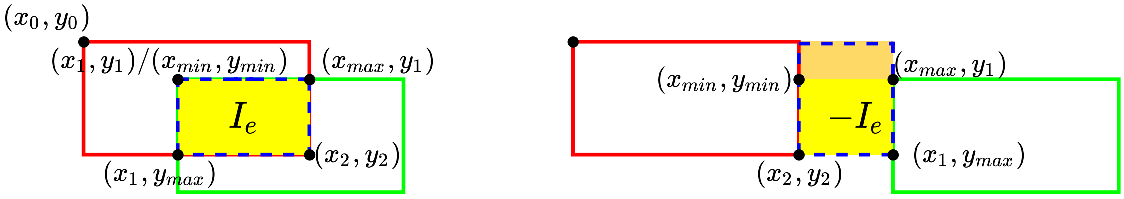
$$\begin{aligned} & x_1 = \max(x_1^t, x_1^p), \qquad && y_1 = \max(y_1^t, y_1^p) \\ & x_2 = \min(x_2^t, x_2^p), \qquad && y_2 = \min(y_2^t, y_2^p) \\ & I_{\mathrm{std}} = (x_2 - x_1)(y_2 - y_1) && \end{aligned} \tag{2-20}$$

那么 IoU Loss 按照如下公式计算：

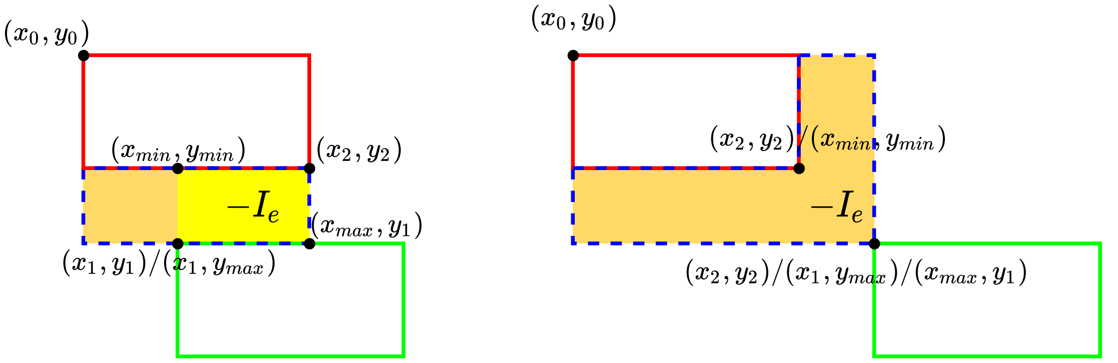
$$L_{\mathrm{IoU}} = 1 - \frac{I_{\mathrm{std}}}{S^t + S^p - I_{\mathrm{std}}} \tag{2-21}$$

IoU Loss 具有尺度不变性，也就是对尺度不敏感 (Scale Invariant)，满足非负性、同一性、对称性和三角不变性，很好的解决了 Smooth-L1 Loss 用于边界框回归的变量相对独立和尺度敏感的问题。但是，当预测框和真实框不相交时，不能反映出两个框的远近，因为无论两框相距多远，其 IoU 值都为 0。另外，IoU Loss 也不能反映预测框和真实框的重合度的大小。因此，本文中采用 Peng 等[52]提出的 EIoU Loss 来替代传统的 IoU Loss。图2-5为 EIoU Loss 在各种重叠方式下的计算示意图，红色框代表目标框，绿色框代表预测框。首先计算图中关键点的坐标值：

$$\begin{aligned} & x_0 = \min(x_1^t, x_1^p), && y_0 = \min(y_1^t, y_1^p) \\ & x_{max} = \max(x_1, x_2), && y_{max} = \max(y_1, y_2) \\ & x_{min} = \min(x_1, x_2), && y_{min} = \min(y_1, y_2) \end{aligned} \tag{2-22}$$



(a)重叠,$(x_1 < x_2, y_1 < y_2)$　　(b)不重叠,$(x_1 > x_2, y_1 < y_2)$



(c)不重叠,$(x_1 < x_2, y_1 > y_2)$　　(d)不重叠,$(x_1 > x_2, y_1 > y_2)$

图 2-5　各种重叠方式下的 EIOU 计算示意图

扩展的重叠区域面积 I_e 为:

$$\begin{aligned} I_e =&(x_2 - x_0)(y_2 - y_0) + (x_{min} - x_0)(y_{min} - y_0) - \\ &(x_1 - x_0)(y_{max} - y_0) - (x_{max} - x_0)(y_1 - y_0) \end{aligned} \tag{2-23}$$

那么 EIoU Loss 按照如下公式计算:

$$L_{\text{EIoU}} = (1 - \frac{I_e}{S^t + S^p - I_e})^2 \tag{2-24}$$

EIoU Loss 将目标框与预测框之间的距离,重叠率以及尺度都考虑进去,解决了 IoU Loss 在两框不重叠时恒为 1.0 的问题,充分反映了预测框和目标框的差距,使得目标框回归变得更加稳定。

2.1.3.3　交叉熵损失函数

交叉熵 (Cross-entropy) 损失函数源于信息论中的信息熵,它用于衡量两个概率分布之间的差异,通常用于分类任务。对于分类问题,假设模型输出的结果为 y,真实标签为 t,则交叉熵损失函数 (CE Loss) 的计算方式如下:

$$L_{\text{CE}}(y,t) = -\sum_{i=1}^{C} t_i \log y_i \tag{2-25}$$

其中，C 表示分类的类别数，t_i 是真实标签中第 i 类的概率，y_i 是模型输出的第 i 类的概率。交叉熵损失函数的作用是用来衡量模型预测的概率分布与真实概率分布之间的差异。当模型的预测与真实标签一致时，交叉熵损失函数的值最小，反之，当模型的预测与真实标签不一致时，交叉熵损失函数的值会增加。因此，交叉熵损失函数被广泛应用于分类问题中，作为优化模型的目标函数，通过梯度下降等优化算法来调整模型参数，使得模型的预测结果与真实标签更加接近。

2.2　人脸检测常用数据集介绍

2.2.1　FDDB 数据集

FDDB(Face Detection Dataset and Benchmark) 数据集[53]是 2010 年发布的一款经典人脸检测基准数据集。该数据集由美国多所大学合作创建，涵盖了各种场景下的人脸图像，共有 2,845 张图像和 5,171 个人脸。这些图像具有不同姿势、遮挡、光照和表情等变化，对人脸检测算法提出了挑战。该数据集采用椭圆形来标注人脸位置和大小，每个人脸由中心坐标、长轴、短轴和旋转角度五个值来描述。该数据集被广泛应用于人脸检测、人脸识别、人脸对齐等领域，并被用于测试和比较多种算法，如 Viola-Jones 算法[13]和 DPM[16-17]算法等。然而，FDDB 数据集使用的椭圆形标签并没有被后续的数据集和基于深度学习的人脸检测器所接受。后续的公开数据集更倾向于使用外接矩形来标注人脸，这样可以通过计算交并比 (IoU) 来简单地定义正/负样本。图2-6展示了 FDDB 数据集的标注方法和部分示例。

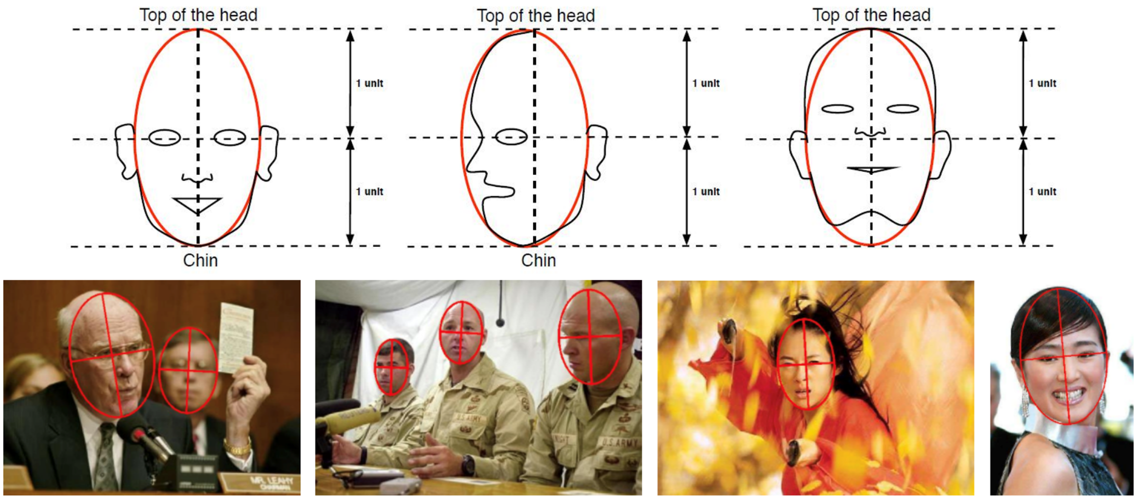


图 2-6　FDDB 数据集的标注方法和部分示例

2.2.2　WIDER-FACE 数据集

2016 年，由香港中文大学和商汤科技共同发布了 WIDER-FACE 数据集[54]，此后迅速取代 FDDB 等公开数据集成为人脸检测领域最具挑战性和权威性的基准数据集之一，许多最先进的人脸检测算法都在该数据集上进行评估和比较。该数据集包含 32,203 张图像，其中包含了 393,703 个标注的人脸框，涵盖了各种实际场景下的人脸。这些图像分为 61 个事件类别，对于每一个事件类别，随机选取其中的 40% 作为训练集，10% 用于交叉验证，50% 作为测试集。每个子集都包含 3 个级别的检测难度：*Easy*，*Medium*，*Hard*。根据实验分析，*Easy* 子集主要是大尺度的清晰人脸标注，*Medium* 子集次之，*Hard* 子集包含所有的人脸标注，并且有很多小尺度或模糊不清的人脸。各子集存在 $Easy \subset Medium \subset Hard$ 的从属关系。WIDER-FACE 数据集具有以下几个特点：

(1) 视觉多样性：包含各种复杂背景、遮挡、姿势和表情等不同情境下的人脸图像，可以用于训练算法的鲁棒性和泛化能力。

(2) 尺度多样性：WIDER-FACE 数据集中的人脸大小变化很大，从小到大覆盖了大多数可能的尺度范围，有助于算法在不同尺度下进行人脸检测。

(3) 人脸数量多：每张图像中可能包含多个人脸，最多可达到 800 个，可以用于评估算法的检测速度和准确率。

(4) 标注精确性：所有人脸框都经过了严格的手工标注，并且提供了四个坐标点来确定每个人脸框位置。

图 2-7　WIDER-FACE 数据集部分示例展示[54]

WIDER-FACE 数据集还推出了评估标准和公开的评估工具，在线提交系统等方便研究者和开发者使用该数据集进行实验和比较算法性能。另外，Deng 等[28]手动添加了人脸关键点标注信息，包括双眼、鼻子和左右嘴角共 5 个关键点，使得该数据集也能用于人脸关键点检测任务。图2-7充分展示了 WIDER-FACE 数据集的特点。

然而，随着深度学习的发展，尤其是更深更复杂的模型被大量提出，人脸检测

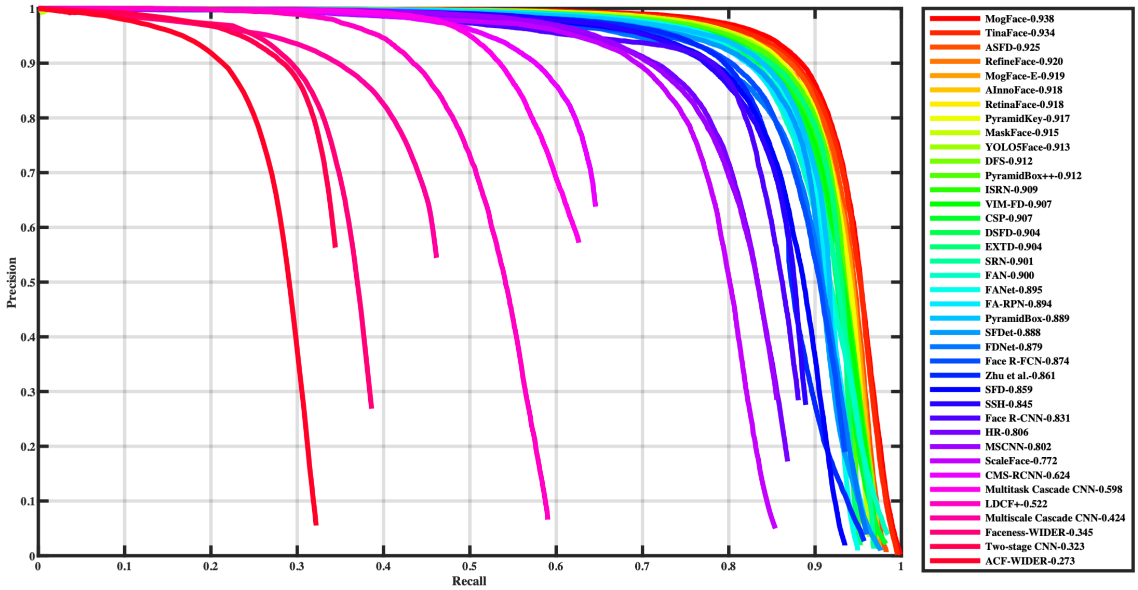


图 2-8　目前 WIDER-FACE 数据集 *Hard* 子集精度排名[54]

任务在 WIDER-FACE 数据集上的精度急剧提升，几乎达到了饱和。图2-8展示了近年来最优秀的人脸检测器在 WIDER-FACE 数据集 *Hard* 子集的评估精度。对最近各年的最高精度进行调查显示[18]，虽然评估精度仍在不断提高，但是趋势已经放缓，并且性能的改进主要来源于模型的膨胀和计算速度的牺牲。因为 WIDER-FACE 数据集不考虑推理速度，很多工作为了提高排名而使用了非常耗时的模型和评估技巧[31,55-56]，这些工作声称具有突出精度却掩盖了检测器性能增益真正来源。相比之下，本文提出的人脸检测器即使在非常困难条件下也能准确地检测到一些人脸，并且速度很快、计算成本很低。

2.3　本章小结

本章简述了深度学习人脸检测技术的理论基础。首先介绍了卷积神经网络的发展历程和基本组成部分 (如卷积层、池化层、激活层、批归一化层等)，并用公式和图表详细说明了标准卷积和深度可分离卷积的运算过程、参数量和计算量计算公式。其次介绍了人脸检测任务常用的损失函数，如 Smooth-L1 损失函数，Cross-entropy 损失函数等，重点介绍了 IoU 损失函数及其改进版 EIoU 损失函数。最后简述了常用的人脸检测基准数据集，包括 FDDB 数据集和 WIDER-FACE 数据集。

第 3 章　YuNet 超轻量级人脸检测算法

3.1　深度学习人脸检测任务概述

人脸检测是计算机视觉的一个基础任务，它的目标是在图像或视频中找出人脸的位置、大小和姿态等信息。近年来，随着深度学习的发展和大规模公开数据集如 WIDER-FACE[54] 的出现，人脸检测技术有了显著的进步。如图3-1所示，深度学习的人脸检测任务主要包括预处理、模型推理和后处理三个步骤。又根据是否更新权重，分为训练和推理两个阶段。图像预处理是对输入的图像进行一些初始化的操作，例如内存重排、缩放和归一化等。

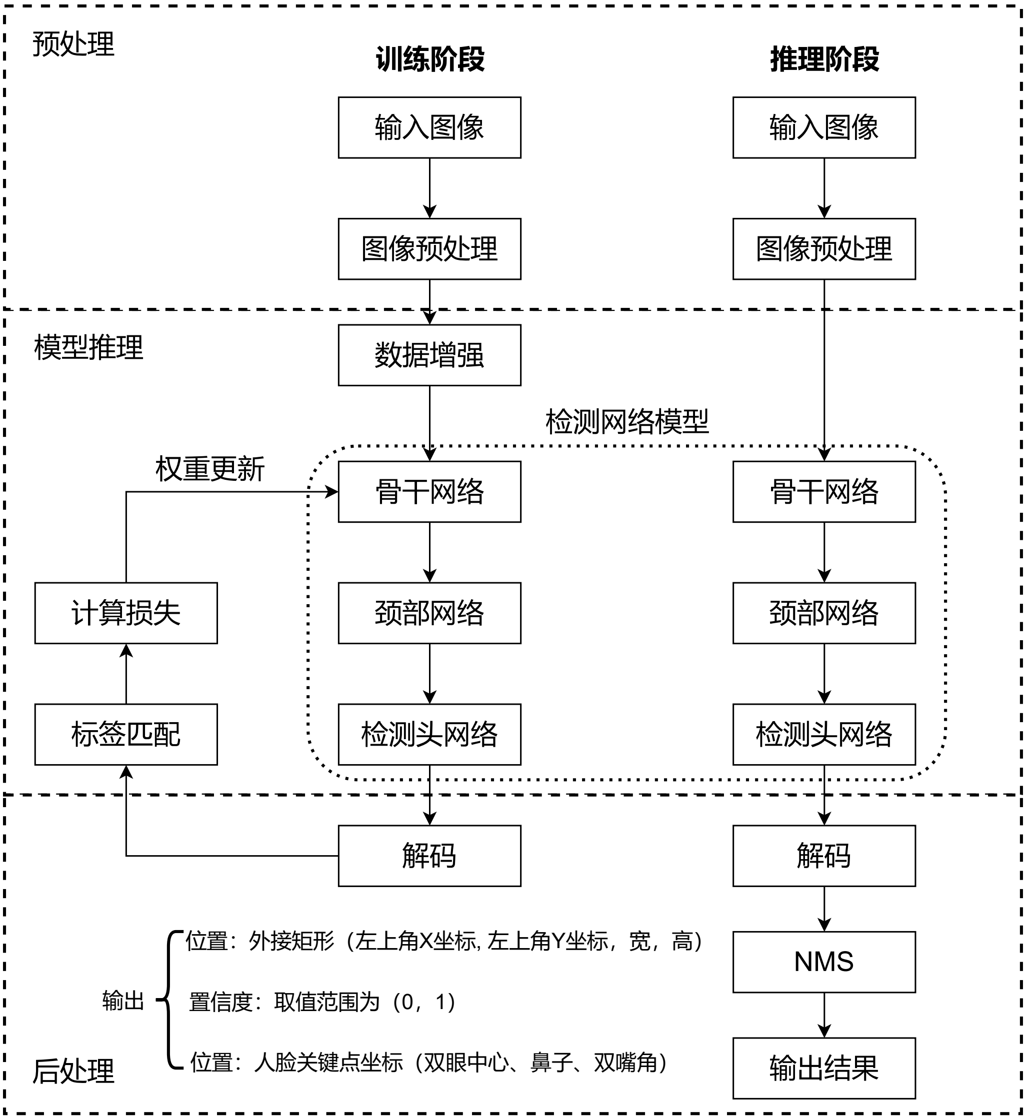


图 3-1　深度学习人脸检测任务的基本流程

人脸检测框架可以根据是否生成 RoI(Region of Interest) 分为多阶段和单阶段两种类型。多阶段的人脸检测器[20,22,57-58]虽然精度高，但是速度慢，无法实现实时 (30 FPS) 检测。单阶段的人脸检测器由于其快速和准确的特点逐渐成为研究热点。单阶段人脸检测模型通常由三部分组成：骨干网络、颈部网络和检测头网络。骨干网络一般采用 ImageNet[39]分类任务中的经典模型如 ResNet[41]系列、VGG[40]系列等，去掉分类层或者做一些微调后作为特征提取器使用。这样做可以利用公开的预训练权重来初始化模型，加快模型收敛，缩短研究周期。颈部网络一般采用特征金字塔网络 (Feature Pyramid Network, FPN)[59]或者其改进版本，如路径聚合金字塔网络 PAFPN[60]、神经网络结构搜索得到的 NasFPN[61]等。这些网络通过建立横向和纵向 (自顶向下、自底向上) 的特征信息流来进行高效特征融合，增强特征上下文信息。

检测头网络设计是整个检测网络设计的核心部分，它包括了网络结构设计和训练阶段的损失函数设计。人脸检测任务可以分解为以下几个子任务：

(1) 人脸分类，目的是将网络输出的边界框划分为前景或背景，其中前景即为人脸，在训练阶段作为正样本计算损失，在推理阶段作为检测结果 (是否是人脸) 输出；

(2) 边界框回归，目的是将预测的前景边界框调整到更接近真实位置；

(3) (可选) 人脸关键点回归，目的是利用额外的人脸关键点标注信息来提高边界框回归的精度，输出的结果也可以用于一些基于人脸属性的高级任务，如人脸对齐等。

根据多个子任务是否共享同一个输出作为分类依据，研究者们将检测头分为共享头和解耦头，并发现在不同的损失函数设计下对检测精度有显著影响。

对于边界框的回归方式，人脸检测方法可以分为基于锚框机制 (Anchor-based) 和无锚框机制 (Anchor-free) 的方法。前者根据先验知识预设多个不同形状和尺度的锚框，例如人脸设置长: 宽 = 1:1，覆盖从小脸到大脸的多个尺度的锚框，模型输出相对于锚框的偏移量。合理的锚框设计能显著提高人脸的召回率和边界框的回归精度。后者通常直接利用分层特征图的步长信息来进行特征图到原图的映射，然后从每个位置的特征点直接回归边界框信息，如 YOLOX[62]回归 (cx, cy, w, h) 信息和 FCOS[63]回归 (l, t, r, b) 信息。论文 ATSS[64]指出，两种方式的差异主要来自于正负样本的定义，也就是说如果训练阶段两者采用相同的正负样本，那么两者最终的性能将十分接近。

3.2　轻量级人脸检测网络结构设计

3.2.1　轻量级网络设计思想

Deng 等[28] 提出的 Retinaface 算法是一种强大的单阶段人脸检测器，它利用联合额外监督和自监督多任务学习，对不同尺度的人脸进行像素级定位，在 WIDER-FACE 数据集上以出色的精度表现取得了当时的 SOTA。如图3-2所示，Retinaface 人脸检测网络由三部分组成，分别为骨干网络、颈部网络和检测头。骨干网络使用 ResNet-50 去除全局池化层和全连接层的剩余部分，后三个 Stage 分别输出 1/8、1/16 和 1/32 输入分辨率的特征层。颈部网络是标准的 FPN 网络，由横向路径和纵向路径聚合而成。横向路径包括一个调整通道维度的 1×1 卷积，和一个融合金字塔特征的融合模块。纵向路径包括一个上采样操作和逐元素相加操作。骨干网络输出的任意特征层首先经过一个 1×1 卷积将通道数统一对齐为 256，然后与上层模块输出的特征图进行逐元素相加并输出到融合模块。检测头网络包括多个串联的特征增强模块和用于输出分类和回归的卷积层。该模型具有 27.3 M 的参数量，在输入分辨率为 320×320 时具有 11.07 GFLOPs 的计算量。骨干网络占据约 86% 的参数量和约 75% 的计算量。

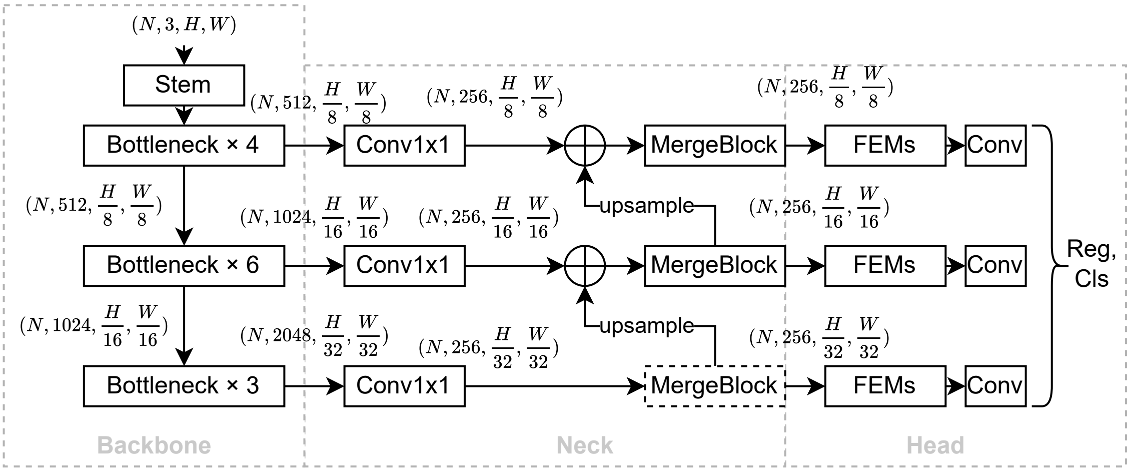


图 3-2　Retinaface 人脸检测器的网络结构示意图

模型参数量是衡量模型复杂度的一个重要的标准，指模型需要训练的参数的总数量，体现了模型的空间复杂度。早期研究人员在进行人脸检测研究时往往会忽视这个因素，因为实验室环境的存储硬件成本低且现代 CPU 的显存够大。但是当面向实际应用时，尤其是部署到一些对成本和速度敏感的边缘设备上时，低参数量不仅能确保运行时的低存储开销，而且能更方便的进行深度访存优化。所以尽可能的降低模型的空间复杂度有着非常重要的意义。对于人脸检测模型，参数量主要体现在堆叠的卷积上。章节2.1.2.1介绍了标准卷积的参数量计算方法。显然，

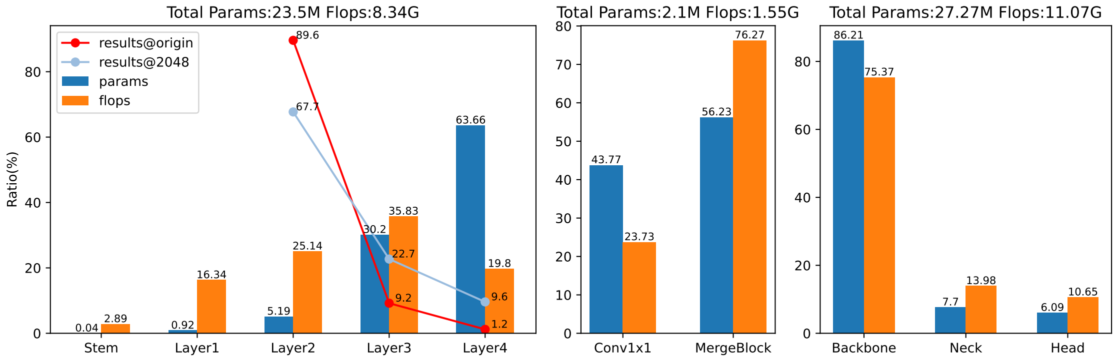


图 3-3　Retinaface[28] 人脸检测网络的参数量和计算量分布

卷积网络的参数量和特征图的分辨率无关，仅与卷积核尺寸和卷积通道数呈正相关。从图3-2左侧可以看到，随着网络深度的加深，卷积通道数是成倍递增的，最顶层 Layer4 甚至高达 1024。图3-3展示了 Retinaface 网络的计算量和参数量分布，左图的折线表示骨干网络不同层输出的候选目标占比。“@2048”指，在测试阶段将测试图片长边放大到 2048，短边等比例缩放。可以看到最顶层的 Layer4 贡献了整个骨干网络近 64% 的参数量，却仅输出了 1.2% 的有效候选目标。同时占总参数量 30.2% 的 Layer3 也仅贡献了不到 10% 的有效候选目标。当然，模型输出的候选目标的统计不仅跟网络各层设计的使命相关，而且跟测试集的目标的尺寸分布有关。我们将测试输入尺寸极端放大到 2048，该尺寸已经远大于人脸检测的常用尺寸，但是 Layer3 和 Layer4 输出的有效候选目标也仅占不到 40%。因此，我们可以肯定在主干网络深层使用巨大的参数量所得收益很小，得不偿失。

所以，对于人脸检测任务，我们有必要对主干网络采取随深度加深通道数激增的设计吗？其实常用的 ResNet，VGG 等主干网络最早设计出来是面向 ImageNet 分类任务的，这种任务不存在尺度多变的问题，只关注最深层并输出 one-hot 的分类结果。这种情况下，随网络深度的增加特征图尺寸会变小，会出现最深层的特征信息严重丢失，因此，研究员往往会通过成倍的增加通道数来保证有效信息的传递。但是在人脸检测中，我们输出有效候选目标数量最多的不是最深层，恰恰相反，是浅层。最深层的作用体现在，能够检测一些中大型目标和通过上采样操作来传递融合的金字塔特征到浅层。因此，直接采用预训练好的用于分类任务的网络来作为主干，尽管能快速的得到比较高的精度，但是整个模型得参数利用效率低，存在巨大的冗余。我们可以总结出两点降低参数量的规律：通道数成倍递增不是必需的，以及尽可能降低模型层数。

模型的计算量是衡量模型复杂度的另一个重要指标，用 FLOPs(Floating Point Operations) 表示，具体指的是模型前向计算中乘法和加法的数量和，体现了模型时间复杂度。章节2.1.2.1介绍了卷积计算量的计算方法。显然，标准卷积的计算量

在卷积的参数量不变的情况下，与输出特征图的尺寸呈正相关。图3-3中 Layer2 的参数量仅是 Layer4 的参数量的 1/12，计算量却更多，主要原因是 Layer2 的特征图单边尺寸是 Layer4 的 4 倍。但是，在卷积参数量固定的情况下，如果我们快速对特征图降维会导致深层网络学习到的特征信息极具减少，极大的降低模型的性能。我们只能在精度满足要求的情况下，尽可能早的对特征图进行降维。虽然我们从模型的角度降低参数量很难，但是我们可以加强模型处理特征的能力，以至于降低输入分辨率也能得到好的评估表现。我们可以总结出降低计算量的规律：减少大尺寸特征图上的卷积运算。

尽管人脸检测模算法的推理时延主要受计算量的影响，但不是唯一因素。ShuffleNetv2[65] 论文中指出，FLOPs 和推理速度不一致主要归结为两个原因：(1) 内存访问成本 (Memory Access Cost, MAC) 不能忽略;(2) 模型的并行程度也影响速度，并行度高的模型速度相对更快。

在小节 2.1.2.2定义下，对于同尺寸的特征图，逐深度卷积和逐点卷积的计算量之比为：

$$\frac{K \cdot K \cdot C_{out} \cdot H \cdot W}{C_{in} \cdot C_{out} \cdot H \cdot W} = \frac{K^2}{C_{in}} \tag{3-1}$$

其中 C_{in} 远大于 K，所以整个模型逐点卷积占有绝大部分计算量。我们可以计算逐点卷积对应的 MAC 为:

$$MAC = H \cdot W \cdot (C_{in} + C_{out}) + C_{in} \cdot C_{out} \tag{3-2}$$

根据均值不等式，当逐点卷积的计算量 $F = C_{in} \cdot C_{out} \cdot H \cdot W$ 为定值时，MAC 存在下限：

$$MAC \geqq 2\sqrt{H \cdot W} \cdot \sqrt{F} + \frac{F}{H \cdot W} \tag{3-3}$$

当且仅当 $C_{in} = C_{out}$ 时成立。所以理论上输入输出通道比为 1 : 1 时卷积的运算速度最快。

逐深度卷积虽然占比较少，但是仍会对速度有一定影响。我们知道逐深度卷积由分组数等于输入通道数 ($g = C_{in}$) 的组卷积实现，其计算量为:

$$F = C_{in} \cdot C_{out} \cdot H \cdot W/g = C_{out} \cdot (H \cdot W)$$

同样的，假设计算量为定值，对应的 MAC 为：

$$MAC = g \cdot \frac{F}{C_{out}} + (C_{out} \cdot H \cdot W + \frac{F}{H \cdot W})$$

可知，MAC 随分组数 g 呈线性正相关，所以在网络设计时应当尽量降低分组数 g。

网络的碎片化会降低模型前向运算的并行度也会造成运算速度减缓。因此我

们在使用一些主干网络中常用的复杂结构如残差结构，多分支聚合结构等时，需要更多的考虑。其实，这些结构主要是用来解决深层网络的退化现象和更好的将浅层特征传递到深层。在实际应用中，轻量级网络卷积层数一般都很少，很难遇到退化现象，而且前文已经说明，使用额外的结构加强浅层特征的传递对结果的影响不大。这些结构却会明显的使网络难以微调，降低推理速度并且降低内存的整体性。所以在设计轻量级网络时，应尽量采取直筒型结构。VGG[40] 风格的网络由 3×3 的卷积密集堆叠和插入最大值池化操作组成，具有速度快、节省内存、灵活性好的特点。所以本文应当采用类似 VGG 风格的网络结构形式，以深度可分离卷积为基本模块进行设计。

3.2.2　网络整体结构

由章节3.2.1的分析可知，面向边缘设备的轻量级人脸检测器的设计应当遵循四点原则：(1) 通道数成倍递增不是必需的；(2) 尽可能降低模型层数；(3) 减少大尺寸特征图上的卷积运算；(4) 降低网络的碎片化。由此设计得到的超轻量级人脸检测网络 YuNet 结构细节如图3-4所示。

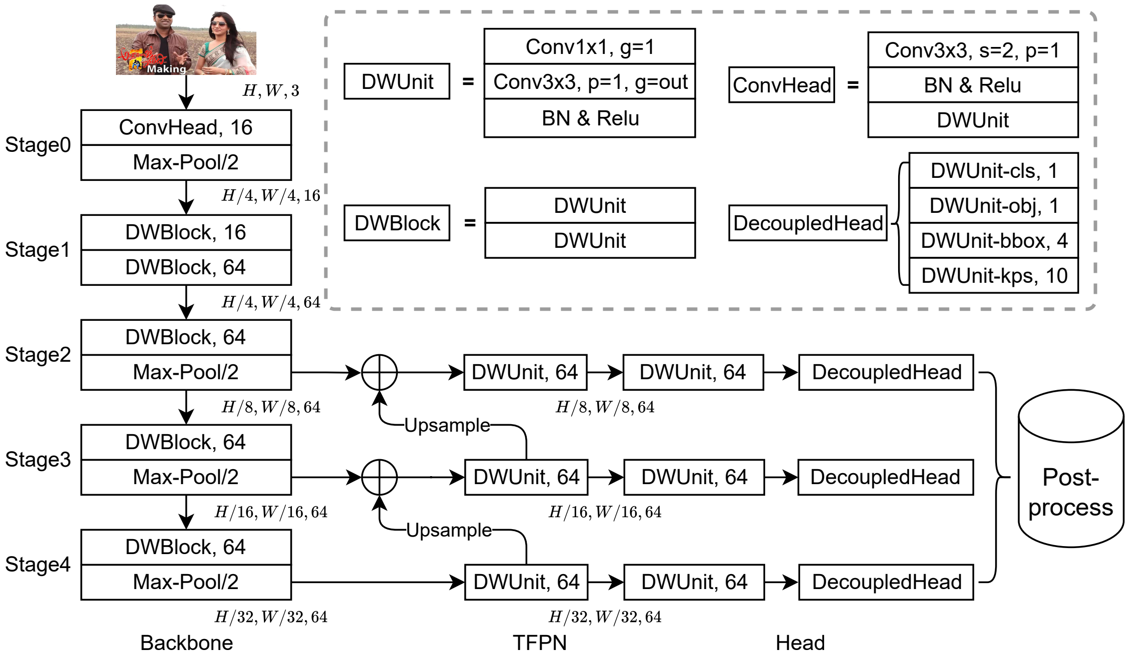


图 3-4　YuNet 网络主体结构

首先介绍网络的基本组成模块。DWUnit 模块是整个网络最基本的模块，它由一个卷积核为 1×1 的逐点卷积、一个卷积核为 3×3 的逐深度卷积、一个批归一化层和一个 ReLU 激活层组成。其中设置卷积步长 1，填充为 1，这样可以保证输出特征图的分辨率不变。当输入输出通道数分别为 C_{in}, C_{out} 时，逐点卷积首先将输出通道数由 C_{in} 提升为 C_{out}，保证接下来的逐深度卷积的输入输出通道数一致。

连续两个 DWUnit 模块组成一个 DWBlock 模块。当 DWBlock 模块输入输出通道数不同时，为了尽可能少的降低参数量和计算量，使用第二个 DWUnit 模块来进行通道数提升。ConvHead 模块由一个步长为 2，填充为 1，卷积核为 3×3 的标准卷积，一个紧接着的批归一化层和 ReLU 激活层，和一个 DWUnit 模块组成。输入 $(H \times W, C_{in})$ 的特征图到 ConvHead 模块，首先经过第一个标准卷积将特征图分辨率降为 $(H/2, W/2, C_{out})$，然后再执行剩下的模块。Maxpooling 模块是 2×2 的最大池化模块，主要用于将输入特征图分辨率降维 1/2。由章节2.1.2.3可知，该模块无需消耗任何参数量，且仅消耗少量计算量，开销几乎可以忽略不计，是最经济简单的降维操作。除开最后的检测头模块，其余网络皆由以上四种模块构成。

表 3-1　骨干网络的计算量和参数量分布细节

Stage i	模块 $\mathcal{F}_i$	特征图分辨率 $H_i \times W_i$	通道数 C_i	参数量 N_i	参数量占比 P_i^p	计算量 (MFLOPs)	计算量占比 P_i^f
0	ConvHead	320×320	16	944	2.21%	24.99	19.81%
	MaxPooling	160×160	-	-		0.41	
1	DWBlock	80×80	64	2,320	28.52%	15.36	63.15%
	DWBlock	80×80	64	9,856		63.90	
2	MaxPooling	80×80	-	-	23.09%	0.41	12.98%
	DWBlock	40×40	64	9,856		15.97	
3	MaxPooling	40×40	-	-	23.09%	0.10	3.24%
	DWBlock	20×20	64	9,856		3.99	
4	MaxPooling	20×20	-	-	23.09%	0.03	0.82%
	DWBlock	10×10	64	9,856		1.00	
				42,688		126.15	

整个骨干网络根据用途被划分为 5 个 Stage，详细的参数量和计算量分布如表3-1所示。依据减少大尺寸特征图上的卷积运算原则，我们在 Stage0 快速的将输入特征图的分辨率降为原来的 1/4，仅使用几乎可以忽略不计的参数量，和约 20% 的计算量。我们在 Stage0 和 Stage1 将输出通道数直接由 3 快速提升到了 64，结合此处特征图分辨率还很高，因此高通道数会导致计算量消耗非常大。有一种选择是，在 Stage1 将输出通道减少，然后在接下来的特征图分辨率更小的地方再将通道数提升。通过实验发现，这样做能显著降低计算量，但是会大幅降低模型表现。原因是少量的计算量无法有效表达输入的高维图片的丰富特征模式[66]。同时，我们在 Stage1 添加了两个 DWBlock 来进行感受野提升，尽管这样做的开销非常大，占约 30% 的参数量和约 65% 的计算量。Stage2-4 是我们的主要输出层，能额外输

出分层特征到颈部网络进行特征融合。依据尽可能降低模型层数原则，每个层先经过一个最大池化层进行快速降维，仅经过一个 DWBlock 模块来提取感受野，然后将特征输出到下一层和颈部网络。依据通道数倍增不是必需的原则，我们将该三层的输入输出通道都固定设置为 64。通过整个骨干网络，我们能得到输入分辨率 1/8，1/16，1/32 的分层特征图，分别提取用于检测大，中，小人脸的粗粒度特征。

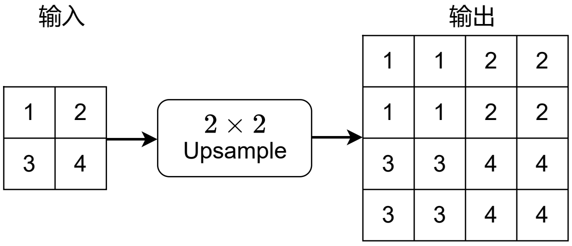


图 3-5　上采样操作示意图

我们使用特征金字塔网络来构建颈部网络。主要分为横向和纵向两条路径，如图3-4所示，经由骨干网络输入的特征图首先经过横向路径进行特征上下文融合，接着经过纵向路径，通过上采样操作加倍特征图的维度，然后和浅层的特征图进行逐元素相加。上采样操作如图3-5所示。我们使用最近邻策略来确定上采样的值，没有任何参数量开销，仅进行少量赋值操作。

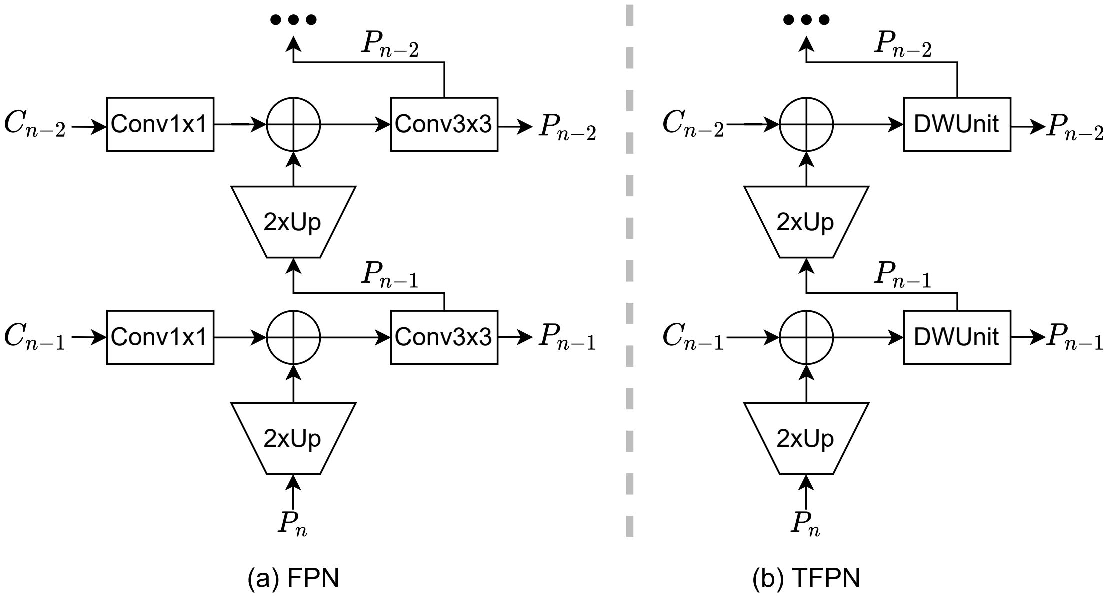


图 3-6　FPN 和 TFPN 的详细实现对比

图3-6是特征金字塔网络的示意图。左边是标准特征金字塔网络 (FPN)，右边是我们提出的极简特征金字塔网络 (Tiny Feature Pyramid Network, TFPN)。两者相比，TFPN 取消了参数量和计算量消耗较大的用于调整通道数的 1×1 卷积层，另

外使用 3×3 的 DWUnit 模块替代 FPN 的 3×3 的标准卷积。我们提出的 TFPN 几乎是 FPN 的各种改进版中最简洁的形式，其中的关键在于取消通道调整卷积。由图3-3可知，在标准 FPN 中，通道调整卷积几乎占有 50% 的参数量和 30% 的计算量。我们能直接取消该卷积的原因是骨干网络已经将输出通道统一为 64，从而不再需要额外的通道调整操作，这是个非常巧妙的设计。后续的消融实验也表明，添加该通道调整卷积除了增加大量开销对精度几乎毫无帮助。表3-2展示了 FPN 和 TFPN 中各组件的开销详情，并添加了骨干网络的对比，定量分析了 TFPN 设计的必要性。

表 3-2　颈部网络各基础组件的开销对比 (输入分辨率为 320×320)

	模块	参数量	计算量 (MFLOPs)
FPN	1x1 通道调整卷积	17,152	9.25
	3 × 3 标准卷积	148,224	78.88
	上采样操作	0	0.0021
	总计	165,376	88.13
TFPN	3 × 3 DWUnit	19,612	10.61
	上采样操作	0	0.0021
	总计	19,612	10.61
骨干网络	-	42,688	126.15

自目标检测的经典网络 Yolov3[67] 和 SSD[68] 的大流行，目标检测网络的检测头大都采用多任务融合的检测头，即分类和回归结果由同一个卷积层进行输出。图3-7展示了多任务融合/解耦检测头的示例。输入一张 (H, W, C_{in}) 的特征图，多任务融合检测头直接输出一个多任务融合张量，然后依据预设定的索引对各任务对应维度进行分割，然后计算损失或经后处理输出结果。实际上这种设计是启发于早期使用全连接层输出结果的分类任务。Ge 等[62] 提出，在训练阶段，将多任务解耦进行分别输出有利于检测网络的快速收敛。由于分类特征属于一些细腻的特征，而回归特征更多是一些轮廓边界特征。如果放在一起，反向传播的时候可能导致网络收敛速度慢，精度降低。通过实验也证实多任务解耦的检测头能加速收敛。因此，本文也采用多任务解耦的检测头。具体做法是针对每种任务设置一个卷积层进行输出。图3-7中有 4 种子任务，因此需要分别设置 4 个具有相应输出通道的卷积层来对输入特征图进行卷积输出。

纵观 YuNet 的网络结构，我们严格执行了卷积网络轻量化的设计原则，将人

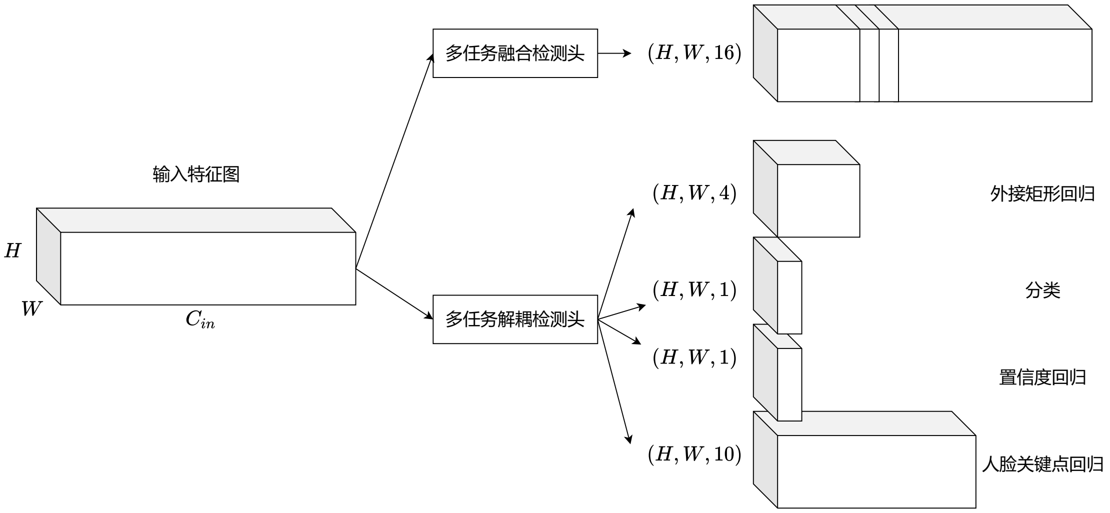


图 3-7　多任务融合/解耦检测头示例

脸检测网络几乎轻量化到极致。整个骨干网络是类似于 VGG[40] 网络的直筒型网络，没有使用 ResNet 的残差连接和 Xception[45] 中的多分支跨层连接结构等，将网络的碎片化降低到最小。同时中间各层模块没有使用例如 MobileNet[42-44] 中的瓶颈结构等对模块内部进行通道放缩的网络结构，而是保证各模块内外的输入输出通道一致，最大化内存访问效率。各层使用最大池化操作进行降维，最大限度的降低降维操作带来的参数量和计算量开销，同时设计上也更加优雅。此外，各层模块的通道数都设置为 8 的倍数，有利于底层实现时的访存优化，尤其是便于适配各种 SIMD 加速指令。

3.3　损失设计

3.3.1　无锚框机制设计

WIDER-FACE 数据集[54] 上排名前列的人脸检测器几乎都遵循原始的基于锚框 (Anchor-based) 的设计模式。然而，基于锚框机制存在许多已知问题。首先，为了达到最优的检测性能，需要在训练阶段开始之前对训练数据集进行聚类分析，确定一组最优的锚框。这些聚类得到的锚框组是严格适用于特定数据集样本分布的，不能在相似任务中通用，阻碍检测算法的迁移 (例如将人脸检测任务的算法迁移到车牌检测任务)。其次，基于锚框机制大大增加了检测头的复杂性，因为输出特征图的每个特征点的预测目标数量和相应的锚框数量呈线性正相关。例如，采用基于锚框机制的 SSD[68] 方法通常在每个特征点上设置 9 个不同的锚框，如果将每个特征点的锚框数量减少至 1 个，那么前者的检测头的参数量和计算量是后者的 9 倍，因为输出通道数是后者的 9 倍。同样，在后处理过程中，前者需要进行处理的预测目标数量也是后者的 9 倍。在一些边缘系统上，在不同设备之间传输如此大

量的预测信息 (例如，从 GPU/NPU 显存到 CPU 内存，因为后处理一般在 CPU 中进行) 可能会成为整体延迟的潜在瓶颈。

无锚框 (Anchor-free) 机制的目标检测器[62-63,69-70]在近几年的发展非常迅速。这些工作表明，无锚框机制的性能完全可以与基于锚框机制相媲美。图3-8是基于锚框机制和无锚框机制在特征图上的示意图。基于锚框机制下特征点预测中心坐标和长宽的偏移值，而无锚框机制下特征点预测中心坐标偏移值和长宽的实际值。采用无锚机制会显著减少检测算法的超参数的数量，大幅降低检测头的复杂度，使检测器的训练和解码阶段变得非常简单。应用无锚框机制的关键在于设计鲁棒的预测框的编码方式以及标签匹配策略。

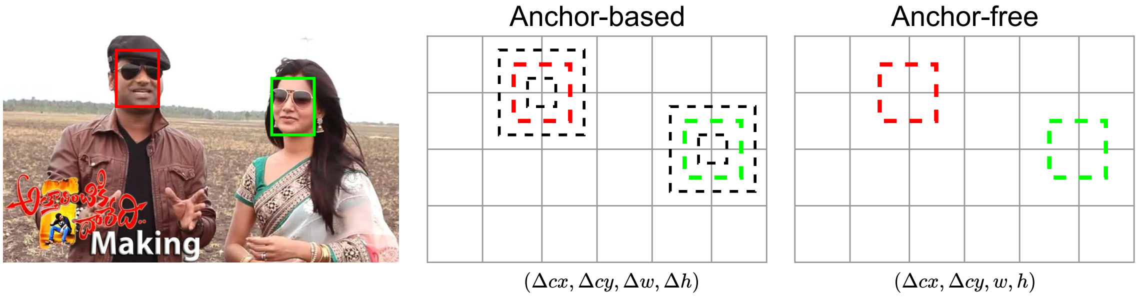


图 3-8　基于锚框机制和无锚框机制在特征图上的示意图

Faster-RCNN[19]目标检测网络提出的基于锚框机制的预测框编码方式启发了后续大量工作，也启发了本文的无锚框机制的编码方式。假定网络输出的特征点的外接矩形框的预测偏移值为 (cx_p, cy_p, w_p, h_p)，在基于锚框的机制下，该特征点所匹配到的锚框为 (cx_a, cy_a, w_a, h_a)，其中 cx, cy 指目标中心点的二维坐标，则该特征点所映射的实际预测目标为:

$$\begin{aligned} cx &= cx_a + cx_p * w_a * var_x \\ cy &= cy_a + cy_p * h_a * var_y \\ w &= w_a + \exp(w_p * var_w) \\ h &= h_a + \exp(h_p * var_h) \end{aligned} \tag{3-4}$$

其中 var 是人为指定的超参数，exp() 指以自然常数 e 为底的指数函数。给定预设锚框尺寸为 (w_a, h_a)，特征图尺寸为 w, h 且降维系数为 $1/s$(即特征图的步长为 s)，则特征图上位于 x, y 的特征点映射到原图上的预设锚框位置坐标为：

$$\begin{aligned} cx_a &= x * w_a / s \\ cy_a &= y * h_a / s \end{aligned} \tag{3-5}$$

显然，特征点映射的实际预测位置直接受预测偏移值和预设锚框决定 (此处忽略超参数 var 的影响)，而预设锚框的位置受特征点的位置和特征图的步长决定。特征

点的位置和特征图的步长都是已知的，那么特征点映射的实际预测位置仅受预设锚框的尺寸 w_a, h_a 和预测偏移值决定。如果我们直接预测尺寸而不是预测尺寸偏移值，则可以将预设锚框这个步骤取消掉，即将检测算法的基于锚框机制改变为无锚框机制。因此，新机制下的特征点映射的实际预测目标为：

$$\begin{aligned} cx &= (x + x_p) * s \\ cy &= (y + y_p) * s \\ w &= \exp(w_p) * s \\ h &= \exp(h_p) * s \end{aligned} \tag{3-6}$$

特征点映射的实际预测人脸关键点为：

$$\begin{aligned} px &= (x + px_p) * s \\ py &= (y + py_p) * s \end{aligned} \tag{3-7}$$

其中 px_p, py_p 是特征点的人脸关键点预测值。

基于锚框机制需要手动设置多个超参数，包括锚框尺寸、var 值，难以调优，且特征点的预测值解码过程计算复杂。而本文提出的无锚框机制不使用任何超参数，特征点的预测值解码过程计算简单，便于优化，更适用于计算能力较差的边缘设备。实际上，通过实验发现，无锚框机制的 YuNet 不仅能取得不低于基于锚框机制的精度，而且模型的鲁棒性和泛化性更强，推理速度更快。

3.3.2 自适应标签匹配策略

在训练阶段，我们需要将真实目标与特征点解码的预测目标按照一定的策略相匹配，匹配上的特征点称为正样本，未匹配上的特征点称为负样本，这个过程称为标签匹配 (Label Assignment)。早期的标签匹配策略是基于预测目标与真实目标的 IoU 值 (交并比，见章节2.1.3.2) 来决定。首先设置 IoU 的阈值 θ，然后对每个实际目标，计算与所有预测目标的之间的交并比，选取 IoU 的最大值，如果该值大于阈值 θ，则确定该值对应的预测目标为正样本。除开所有正样本，剩下的所有预测目标均为负样本。由于每个实际目标匹配仅至多一个预测目标，所以正样本的总量很少，而剩余的负样本数量却非常庞大。正负样本数量失衡会导致网络难以收敛，所以 Shrivastava 等[71]提出在线困难样本挖掘 (Online Hard Example Mining, OHEM) 算法对所有负样本进行过滤，选出与正样本数量匹配的困难负样本。尽管在一定程度上解决了负样本数量庞大的问题，使目标检测网络能顺利收敛，但是更有效的标签匹配策略仍受广大研究者热衷。近几年目标检测任务的标签匹配取得了重要进展[62-64,72-73]，极大的提高了检测精度。一个优秀的标签匹配策略主要

有四个关键点[62]：(1) 基于检测网络的预测值来确定与实际目标的匹配关系；(2) 限制匹配的预测目标中心点落在实际目标的一定范围内；(3) 对每个实际目标选取动态数量的正样本；(4) 根据全局信息来确定预测目标所匹配的实际目标。

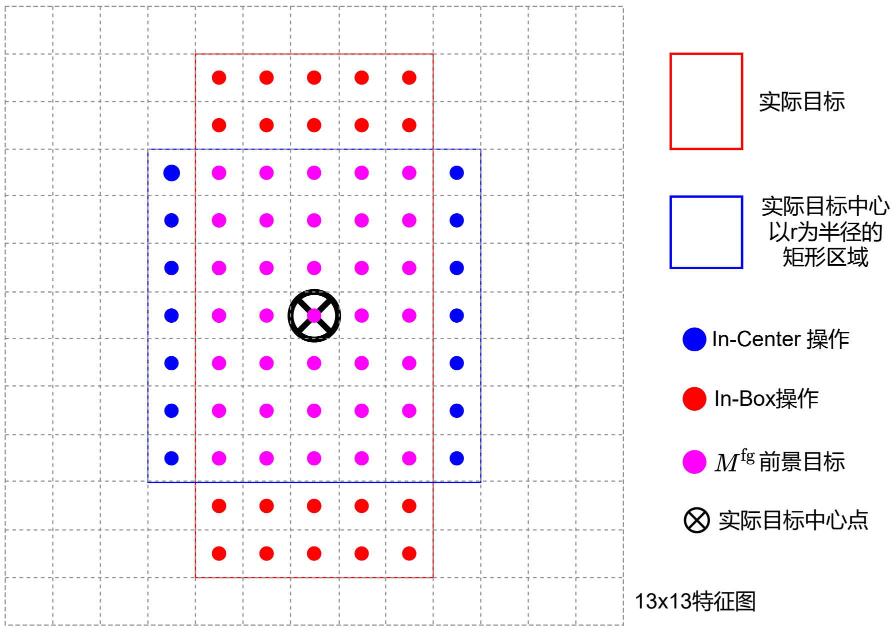


图 3-9　In-Box 操作和 In-Center 操作示意图

具体而言，结合伪代码3-1，本文研究的自适应标签匹配策略可以归纳为四个步骤：

(1) 在特征图上筛选出前景特征点集合。筛选过程如图3-9所示，红色矩形框是实际目标所占的区域，蓝色方形框是在实际目标中心往外扩展 r 个单位所占的区域，简称为辐射区域，r 是超参数需提前手动设置，本文取值 3.0。首先进行 In-Box 操作筛选出落在实际目标区域内的特征点集合，即红色点集合。接着进行 In-Center 操作筛选出落在辐射区域内的特征点集合，即蓝色点集合。对两集合取交集得到前景特征点集合，即粉色点集合。

(2) 计算前景特征点集合与实际目标集合之间的成本矩阵。第 i 个前景特征点与第 j 个实际目标的成本为:

$$C_{ij} = L_{ij}^{\mathrm{cls}} + \gamma L_{ij}^{\mathrm{reg}} \tag{3-8}$$

其中 L^{cls} 指预测目标置信度的分类损失，一般使用二元交叉熵损失 (BCELoss) 来计算。L^{reg} 指预测目标位置的回归损失，一般使用 IoULoss 来计算。γ 是用于调节分类损失与回归损失之间权重的超参数，本文取值 3.0。

(3) 对每个实际目标动态匹配 k 个正样本。首先对每个实际目标，计算它与所有前景特征点的成本和，记为 S，对 S 进行向下取整就得到 k，如果 k 等于 0，则表示这个实际目标没有匹配上任何前景特征点，这种情况是禁止的，所以这种情

况下直接给 k 取值 1，保证所有实际目标都能匹配至少一个前景特征点。

(4) 按照成本值从小到大选择 k 个前景特征点进行匹配。如果一个前景特征点被多个实际目标匹配上，则保留成本小的匹配，取消成本大的匹配。最后，将匹配上的前景特征点作为正样本，剩余未匹配的前景特征点作为负样本。匹配完成，输出匹配结果。

根据以上步骤，我们基于网络的预测值动态的得到了所有的正负样本。这个过程中仅有两个超参数 r, γ，且模型表现对两者的取值波动不敏感。另外，我们的检测头会同时输出人脸关键点预测，但是人脸关键点预测不参与标签分配的计算。

算法 3-1　自适应标签匹配策略

```
Input:
    $I$ 指输入图片;
    $A$ 指预设锚框的集合;
    $G$ 指输入图片 $I$ 的实际目标集合;
    $r$ 指中心区域范围系数;
    $\gamma$ 指计算成本时的回归损失系数;
Output:
    $\pi$ 指正负样本匹配集合
$P^{\mathrm{cls}*}, P^{\mathrm{box}*} \leftarrow$ Forward($I, A$);
$M^{\mathrm{fg}} \leftarrow$ In-Box($P^{\mathrm{box}*}, G$) $\cup$ In-Center($P^{\mathrm{box}*}, G, r$)
$P^{\mathrm{cls}}, P^{\mathrm{box}} \leftarrow$ Filter($P^{\mathrm{cls}*}, P^{\mathrm{box}*}, M^{\mathrm{fg}}$);
pairwise $cls$ cost: $C^{ij}_{\mathrm{cls}}$ = BCELoss($P^{\mathrm{cls}}_j, G^{\mathrm{cls}}_i$)
pairwise $reg$ cost: $C^{ij}_{\mathrm{reg}}$ = IoULoss($P^{\mathrm{box}}_j$,$G^{\mathrm{box}}_i$)
$bg\ cls$ cost: $C^{\mathrm{bg}}_{\mathrm{cls}}$ = CELoss($P^{\mathrm{cls}}_j, \emptyset$)
$fg$ cost: $C^{\mathrm{fg}} = C_{\mathrm{cls}} + \gamma C_{\mathrm{reg}}$
final cost matrix: $C \leftarrow$ concatenating $C^{\mathrm{bg}}_{\mathrm{cls}}$ to the last row of $C^{\mathrm{fg}}$
for $i = 0$ to $|G|$ do
    dynamic $k_i \leftarrow$ Max(Ceil(Sum($C_i$)), 1)
    $\pi^*_i \leftarrow$ choosing $k$ matchers for $G_i$
end
$\pi \leftarrow$ Global-View($\pi^*$)
return $\pi$
```

3.3.3 多任务联合损失设计

本文提出的 YuNet 人脸检测网络支持人脸检测以及人脸关键点检测，是一种多任务的人脸检测器。多任务解耦检测头预测四种子任务，分别是：外接矩形回归，分类，置信度回归，和人脸关键点回归。在训练阶段，我们最小化多任务损失：

$$L = \frac{1}{N}\sum^{N}(L_{cls} + \alpha_1 L_{obj} + \alpha_2 L_{bbox} + \alpha_3 L_{ldm}) \tag{3-9}$$

其中 N 表示章节3.3.2得到所有正样本的数目，超参数 α_1, α_2 and α_3 是用来平衡各子任务损失的权重系数，本文分别设置为 1.0, 5.0 和 0.1。L_{bbox} 为人脸外接矩形框的回归损失，采用 EIOU 损失来进行计算。L_{ldm} 为人脸关键点回归损失，采用 Smooth-L1 损失来进行计算。L_{cls} 为人脸分类损失，采用交叉熵损失来进行计算。L_{obj} 为置信度预测损失。许多之前的工作，如 Yolov3[67] 中提出的 objectness 分支，FCOS[63] 中提出的 centerness 分支等，都是为了缓解单阶段目标检测的分类得分和定位精度之间不匹配的问题。同样，本文同样添加一个 objectness 分支来单独输出一个置信度，对所在特征点是否为正样本进行打分。在训练阶段我们使用交叉熵损失计算 L_{obj}。在推理阶段，首先将网络输出的分类预测值和置信度预测值都通过 Sigmoid 算子将值域调整为 $(0, 1)$，然后输出检测置信度为:

$$\mathrm{conf} = \sqrt{P_{obj} \cdot P_{cls}} \tag{3-10}$$

所涉及各种损失的详情及特点见章节2.1.3。

3.4　轻量级模型训练策略设计

本文为了实现模型轻量化，采取了减少模型中的参数数量、减少层数、降低分辨率等操作，得到极致的轻量化模型。但是极致的轻量化会导致人脸检测模型的感知能力大幅降低，弱化对人脸关键特征的识别能力，从而造成检测精度降低。过低的检测精度是不可接受的，会使得轻量化的工作变得毫无意义。传统的基于 Adaboost 的 V-J 检测器[13] 尽管检测速度非常快，也是由于检测精度过低糟致弃用。当然我们在章节3.3提出的无锚框机制和相应的自适应标签匹配策略以及多任务联合损失，通过训练过程的强监督提高模型的鲁棒性和泛化能力，在很大程度上缓解了精度降低问题。实际上，除了模型架构设计和训练过程的影响，人脸检测精度还与数据集质量强相关。本小节首先讨论 WIDER-FACE 数据集的样本尺寸分布，然后对训练阶段常用的数据增强算法进行分析，提出样本均衡的数据增强算法。

3.4.1　WIDER-FACE 数据集分析

章节2.2详细介绍了 WIDER-FACE 数据集的特点：视觉多样性、尺度多样性、人脸数量多和精确标注。其中尺度多样性是制约数据集的评估精度最主要的因素。WIDER-FACE 数据集不仅存在仅有几个像素的小脸，也存在几十万个像素的大脸，从小到大覆盖了绝大多数现实场景中可能出现的尺度范围，有助于提升人脸检测模型在不同尺度下的检测能力，也给模型的训练带来挑战。我们对 WIDER-FACE 训练集的所有人脸标注的尺度进行了统计分析，所有人脸的尺度分布如图3-10所

示。横坐标为各人脸矩形框的平均边长由如下公式计算：

$$s = \sqrt{H \cdot W} \tag{3-11}$$

其中 H, W 为人脸矩形框的长宽，s 为人脸平均边长，这里称为尺度。纵坐标为人脸尺度的累计百分比，表示低于某尺度的人脸数量占总人脸数量的比值。“Origin Scale”线，即蓝色线，表示不对数据集图片做任何缩放变换情况下的原始人脸尺度分布。“Long Scale = 640”线，即黄色线，表示将人脸图片的长边缩放到 640，短边保持纵横比缩放，不足 640 的部分进行零填充，然后对缩放后的人脸通过公式3-11计算得到的人脸尺度分布。

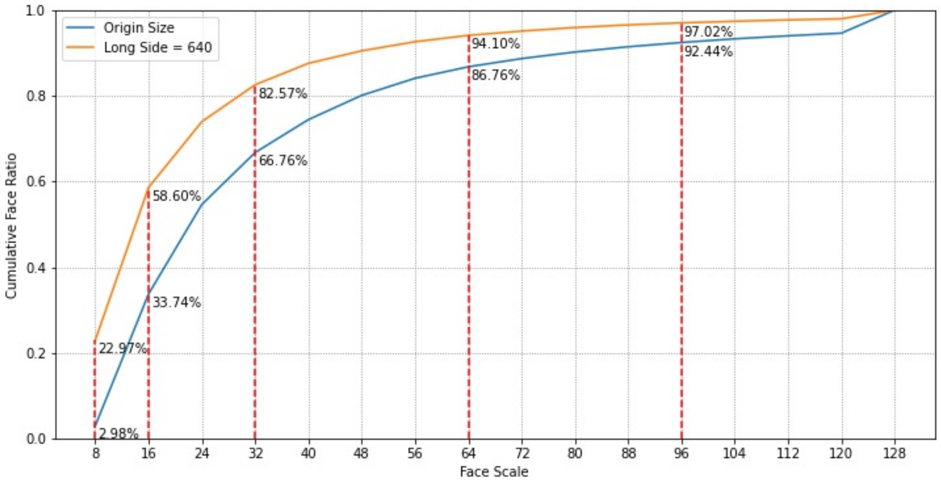


图 3-10　WIDER-FACE 训练集人脸尺度分布

在训练阶段，为了降低数据 IO 对训练速度的限制，我们通常以批次 (batch) 的形式将多张训练图片同时进行输入。输入张量需要维度对齐，那么要求同时输入的图片为同一尺寸，例如 640 × 640，而训练集中的图片都是随意搜集的包含多样尺寸的图片，因此需要对每张输入图片进行尺度缩放以满足要求。然而所述的尺度缩放操作伴随着整个样本尺度分布的剧烈变化。图3-10中显示，当将原始尺寸的图片直接缩放到 640 时，边长小于 8 像素的极小人脸由 2.98% 增加到 22.97%，意味着小尺寸人脸的数量显著增加。此外，缩放后 82.57% 的人脸边长将小于 32 个像素，而原图中该数值仅为 66.76%。大量的小尺寸样本会在计算损失时因尺寸太小会破坏标签匹配策略的合理性从而被丢弃，限制了样本多样性。大尺寸样本数量的骤减也会降低模型对大脸检测精度。同时，样本尺度分布的剧烈变化会导致测试和训练的样本尺度分布不一致，例如，模型在缩放后的 640 × 640 分辨率下进行训练，而在原始分辨率上进行评估，会整体影响模型的评估精度。

3.4.2　常用数据增强方法分析

人脸检测任务的数据增强是一种通过对原始人脸图片数据进行变换和扩充的技术，以提高模型性能和泛化能力的方法。数据增强可以模拟不同的场景和情况，从而增强模型的鲁棒性，减少过拟合的风险。常用的数据增强算法有：Mosaic、RandomFlip、PhotoMetricDistortion 以及 RandomCrop 等[74]。图 3-11展示了相关常用数据增强算法的可视化效果。Mosaic 算法在 Yolov4[75]算法上表现出色，是目标检测任务中常用的扩充样本多样性的数据增强算法。该算法每次从训练数据集选择 4 张图片 (通常数量为 4 或 9)，然后对这 4 张图片进行缩放翻转等变换，再拼接获得一张新的图片，同时组合所有的标注，过滤范围外的标注。这种数据增强算法能极大的丰富检测物体的背景，并大量增加小目标的数量。RandomFlip 算法，即随机翻转算法，在训练开始前会人工指定一个翻转概率阈值，通常设为 0.5。在每次加载图片时，算法随机生成一个翻转概率，如果低于阈值则执行翻转操作。由于自然人脸是近似左右对称的，且训练图片中的人脸大多是正常双眼在上嘴巴在下的状态，所以仅进行水平翻转。通过翻转增强算法能丰富训练样本的多样性。PhotoMetricDistortion 算法，即照片颜色对比度增强算法，会随机调整图像的颜色通道值，例如对亮度、对比度、饱和度等进行调整，从而增加训练样本的多样性。RandomCrop 算法，即随机裁剪算法，是各种检测任务中最常用的数据增强算法，具体操作流程如图3-12所示。在训练阶段开始前，人工指定一个离散的尺度缩放值的集合。每次加载图片时，随机从缩放值集合中选择一个缩放值，然后根据图片


(a) 原图
(b) RandomFlip
(c) PhotoMetricDistortion
(d) Mosaic
(e) RandomCrop

图 3-11　四种人脸检测常用的数据增强方法效果

尺寸缩放得到裁剪区域尺寸。接着根据裁剪区域大小随机一个合理的左上角坐标，最后对随机定位到的图像区域进行裁剪并缩放到指定训练尺寸。显然，该数据增强算法会显著的扩充数据集的样本数量以及背景的多样性，同时改变样本的尺度分布。通过实验发现，该算法的评估表现对尺度缩放值集合的设置非常敏感，设计一种有效的尺度缩放值集合选择策略非常重要。

3.4.3 样本均衡的数据增强算法设计

对于极致轻量级的人脸检测网络模型，很多在大模型上常用的且取得出色效果的数据增强算法并不适用甚至取得精度降低的反效果。例如，我们通过在 YuNet 模型上的实验发现，采用 Mosaic 算法会造成训练速度极慢且评估精度的大幅下降，约 11%。我们猜测原因是小模型对人脸关键特征的感知能力太差，几乎无法从 Mosaic 算法通过增强扩充的复杂人脸特征中获得收益，甚至被稀释了感知能力。采用 PhotoMetricDistortion 算法会导致精度下降，但是降幅很小，约 1%。实际上，应用改变样本尺度分布的数据增强算法相比保持样本尺度的算法，得到的评估精度波动大很多。因此，对于极致轻量级的模型，研究适合的改变尺度分布的数据增强算法是最有效的评估精度提升手段。进一步研究得知，保持训练/测试阶段样本尺度分布一致是取得最佳评估精度的关键所在。最终，我们设计了 RandomCrop 和 RandomFip 算法组合的样本均衡的数据增强算法。

图 3-12 RandomCrop 数据增强算法示意图

常见的 RandomCrop 算法的尺度缩放值集合为在 [0.3, 1.0] 区间的所有间隔为 0.1 的离散值组成的集合。Guo 等[36] 为了增加小样本的数量，将上述区间范围扩展为 [0.1, 3.0]。当尺度缩放值大于 1.0 时，裁剪区域将会包含整个输入图像，图像之外的区域使用零值进行填充。另外，他们提出了一种可搜索的放大缩小尺度空间，并以 WIDER-FACE 验证集上的精度为优化目标来构建最佳的尺度缩放值集合。显然，只要有足够多的轮次来枚举所有的尺度缩放值的组合，总能找到在当前测试条件下的最优集合。但是，在实际场景中的样本尺度分布本身是不可知的，在测试

集上根据评估精度进行优化迭代也不符合算法研究的规范。实际上，我们仅能低成本知晓的是训练集的样本尺度分布。因此，我们将对尺度缩放值集合的优化由受测试集上的评估精度监督，转化为优化随机裁剪之后的样本尺度分布与训练集原始图像中的样本尺度分布相接近的集合。这样做有效的前提是，我们默认训练集和测试集的原始样本分布是一致的，于是可以近似的用训练集的样本分布来模拟测试集的样本分布。本文采用图3-13所示的样本尺度分布图来描述各分布的接近程度。横坐标为样本尺度，由公式3-11计算得到。纵坐标为各尺度的数量占比。在训练阶段之前，本文采用如下步骤确定最佳尺度缩放值集合：

(1)画出训练集原始尺寸的样本尺度分布。

(2)确定初始样本缩放值集合为 [0.3, 2.0]，并设置为当前集合。

(3)遍历训练集，应用以当前集合为参数的 RandomCrop 算法，统计并画出当前集合下的样本尺度分布。

(4)比较当前分布与原始分布的图形接近程度，如果比较接近则记录为候选集合。如果峰值偏左上则减少集合右侧的离散值，如果偏右下则减少集合左侧的离散值。将新得到的集合设置为当前集合，并迭代执行步骤 (2)。

(5)选择最接近的 2-3 个候选集合进行完整的训练，对比评估精度。

由于步骤 (3) 仅需要遍历整个训练集的标注，不需要实际读取图片，所以效率非常高。相比于 Guo 等[36]需要大批量完整训练，本策略的开销几乎是可以忽略不计的。图3-13中红色曲线是原始样本尺寸分布曲线，其他颜色线是候选样本尺度分布曲线。显然，天蓝色线即“crop_05_15”线最接近原始曲线，表示 [0.5, 1.5] 的尺度缩放集合是当前最优的。

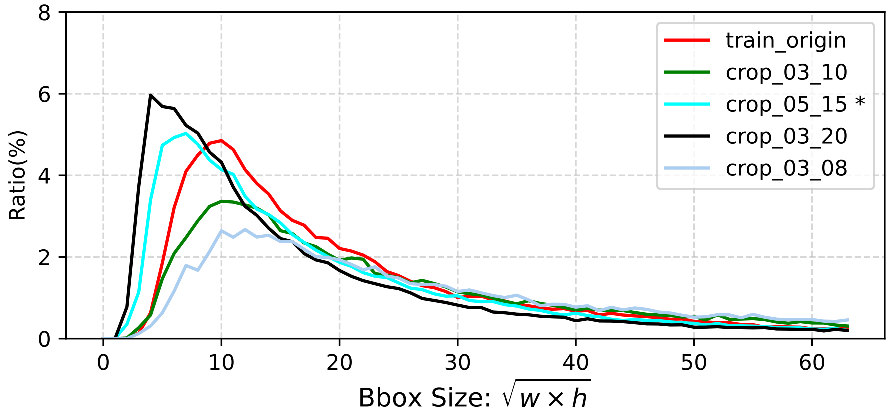


图 3-13　人脸尺度分布曲线

3.4.4　训练设置

为了更有效的组织实验，本文使用开源的 PyTorch 和 MMDetection 框架[76]来实现 YuNet。本文采用 SGD 作为优化器，其中参数 momentum 为 0.9，参数 weight-decay 为 0.0005。使用两张 NVIDIA 1080Ti (12 GB) 显卡进行训练，batch 大小设置为 16。在前 1500 次迭代中将学习率由 0.001 线性增加到 0.01。之后采用 StepLR 调度器来调整学习率，分别在第 400 个轮次和第 544 个轮次将学习率衰减为前值的 1/10。在没有任何预训练的情况下，YuNet 可以在 560 个轮次内从零训练完毕。

3.5　实验设计

3.5.1　实验设置

为了从模型复杂度、推理延迟、检测精度等方面进行公平且综合的比较，本文按照三个准则来收集用于对比的轻量级人脸检测算法。(1) 模型参数量较小，计算量在 640 × 640 输入分辨率下小于 1 GFLOPs；(2) 原作者已经公开发布了源代码并提供训练好的模型权重；(3) 算法来源于影响力大的会议、期刊或者开源项目。我们筛选出了以下先进的人脸检测器来作为我们的对比对象，它们分别是 SCRFD[36]，Retinaface[28]，和 YOLO5Face[34]。一些其他工作如 Blazeface[77](缺乏官方发布的代码)，FaceBoxes[32](缺乏官方发布的代码，精度相对较差)，LFFD[33](计算量太大，精度相对较差) 被排除。此外，我们注意到这些算法在其论文中描述的结果并不是在统一的测试条件下进行评估的，这会极大地影响对比结果。所以，为了进行公平的比较，我们对实验环境做如下约定：

1) 所有用于对比的模型都被转换为 ONNX 格式，然后使用 ONNXRuntime 在 CPU 下进行推理。我们参照官方源代码，用 NumPy 重新实现了这些方法的数据预处理和后处理。

2) 不使用任何测试增强 (Test Time Augmentation, TTA) 技巧例如：图像翻转、多尺度预测、模型集成等，我们分别使用 320 × 320、640 × 640 和原始尺寸 (大约为 1000) 作为输入分辨率，在 WIDER-FACE 验证集上进行评估。我们将置信度阈值设置为接近零 (例如 0.01)，以获得最佳的 mAP，尽管这种操作会导致大量的误检，但是能最大限度的展示模型的检测能力。NMS 阈值固定设置为 0.45。

3) 由于大多数移动和嵌入式设备由于硬件限制只有 CPU 而没有 GPU，我们只评估在 CPU(Intel i7-12700K) 下的推理效率。推理延迟结果是由测试整个 WIDER-FACE 验证数据集的总时间 (除开加载图片和计算 mAP 的时间) 取平均值得到的。计算量和参数量的统计方法来自 Feng[18]。

另外，我们将 YuNet 的 Stage1 的通道数减半，并将多任务解耦检测头前用于融合特征的 DWUnit 减少一层，得到进一步轻量化的 YuNet-s。本文也将 YuNet-s 加入对比清单。为了体现与最先进人脸检测器的评估精度差距，我们添加了与大模型[28,36]的评估对比。

3.5.2　实验结果分析

表 3-3展示了各个评估指标下 YuNet 和其他相似方法在 WIDER-FACE 验证集上的评估结果对比，最好的结果用下划线标出。其中 YOLO5Face 由于其导出的 ONNX 模型不支持动态分辨率输入，不参与原始分辨率下的比较。另外，YuNet-s、Retinaface-n、SCRFD-10g 不参与指标数值的比较。从表中可以看出，本文的 YuNet 只有 75,856 个参数，比其他轻量级模型少了一个数量级。YuNet 的计算量也是最小的，在 640 × 640 分辨率下只有 595 MFLOPs。在推理时延方面，YuNet 也远远快于其他轻量级模型，在所有测试分辨率下都能实时运行 (30 FPS)，而且

表 3-3　YuNet 和其他相似方法在 WIDER-FACE 验证集上的评估结果对比

Input Size	Methods	#Params(ratio)	#FLOPs (M)	AP_{easy}	AP_{medium}	AP_{hard}	Latency (ms)
320 × 320	SCRFD-0.5g (ICLR22)[36]	631,410(8.32x)	195	0.850	0.754	0.372	3.4
	Retinaface-0.25 (CVPR20)[28]	426,608(5.62x)	245	0.765	0.611	0.271	4.2
	YOLO5Face-n (CVPR22 Workshop)[34]	446,376(5.88x)	185	$\underline{0.858}$	$\underline{0.793}$	$\underline{0.445}$	7.2
	YuNet(ours)	$\underline{75{,}856(1.00x)}$	$\underline{149}$	0.836	0.747	0.395	$\underline{2.2}$
	YuNet-s(ours)	54,608(0.72x)	96	0.785	0.668	0.309	1.9
	Retinaface-n	27,293,600(359.81x)	11,070	0.868	0.742	0.341	49.1
	SCRFD-10g	4,229,905(55.76x)	3,359	0.923	0.862	0.504	17.3
640 × 640	SCRFD-0.5g	-	779	$\underline{0.907}$	$\underline{0.882}$	0.684	17.8
	Retinaface-0.25	-	981	0.893	0.831	0.541	22.0
	YOLO5Face-n	-	741	$\underline{0.907}$	0.880	$\underline{0.734}$	20.1
	YuNet(ours)	-	$\underline{595}$	0.899	0.869	0.691	$\underline{11.3}$
	YuNet-s(ours)	-	386	0.876	0.834	0.591	8.7
	Retinaface-n	-	44,260	0.943	0.908	0.659	232.7
	SCRFD-10g	-	13,435	0.949	0.935	0.814	95.0
Origin Size	SCRFD-0.5g	-	-	0.892	$\underline{0.885}$	$\underline{0.820}$	25.0
	Retinaface-0.25	-	-	$\underline{0.907}$	0.883	0.742	57.0
	YuNet(ours)	-	-	0.892	0.883	0.811	$\underline{16.3}$
	YuNet-s(ours)	-	-	0.887	0.871	0.768	13.8
	Retinaface-n	-	-	0.955	0.941	0.847	463.7
	SCRFD-10g	-	-	0.923	0.925	0.885	137.8

输入分辨率越大，速度优势越明显。在评估精度方面，YuNet 在原始分辨率下达到了 0.892/0.883/0.811 的精度，略低于 SCRFD-0.5g，在 *Medium* 和 *Hard* 子集上高于 Retinaface-0.25；在 640 × 640 输入分辨率下达到了 0.899/0.868/0.691 的精度，仍略低于 YOLO5Face-n，和 SCRFD-0.5g 相当，在 Retinaface-0.25 之上；在 320 × 320 输入分辨率下达到了 0.836/0.747/0.395 的精度，略低于 YOLO5Face-n，在 *Hard* 子集上高于 SCRFD-0.5g，在 Retinaface-0.25 之上。虽然 YuNet 没有在所有对比算法中都获得最高的评估精度，但 YuNet 具有极小的参数量和极快的速度，这是其他方法所不具备的。而且 YuNet 和规模百倍于其的大模型相比，并没有产生明显的精度差距，但是在推理速度上却有数量级的优势。另外，本文并未将 YuNet 与对大模型进行剪枝压缩或者量化后的模型进行对比，可能会弱化大模型的性能表现。原因是对大模型剪枝和量化操作带来的参数量和计算量降低无法弥补其与 YuNet 的数量级的差距，而对超轻量级模型的剪枝和量化操作会导致检测精度大幅降低。同时，对大模型的剪枝压缩和量化等操作需要进行大量的额外处理，也可能导致模型部署时对推理框架产生依赖，显著增加模型的落地部署的难度，这也突出了 YuNet 部署简单的优势。

YuNet-s 是基于 YuNet 进一步减少计算量和参数量的版本，计算量减半，参数量减少 30%，速度大幅提升，评估精度却没有明显降低。对于计算和存储资源非常有限的边缘设备来说，本文建议使用 YuNet-s 部署人脸检测任务，既能保证精度要求又能享受极致的计算性能。

表 3-4　YuNet 与传统基于手工特征人脸检测方法对比

方法		Viola-Jones[13](传统方法)		YuNet		YuNet-s	
存储开销 (float32)		908.3 KB		296.3 KB		213.3 KB	
		时延 (ms)	FPS	时延 (ms)	FPS	时延 (ms)	FPS
分辨率	224 × 224	2.5	395	1.1	909	0.9	1,111
	320 × 320	5.8	172	1.6	625	1.4	714
	640 × 480	10.7	94	9.6	104	7.8	128
	1280 × 960	41.2	24	36.2	28	29.3	34

表3-4展示了 YuNet 与基于手工特征的传统方法在模型存储开销和各种常用分辨率下的推理速度对比。实验时保持测试条件不变，除了在同一幅图像上循环测试 1000 次而不是整个 WIDER-FACE 验证集。对比实验中，我们采用典型的基于 Haar 特征的 Viola-Jones[13]算法来代表传统方法，算法实现来自开源 OpenCV 库。为了公平对比，我们尽可能实现了和 YuNet 一致的后处理方法。Viola-Jones 的存储开销来自其训练后保存的模型 xml 文件，YuNet 的存储开销来自 float32 表示的

模型权重。可以看到，YuNet 的存储开销仅约占 Viola-Jones 的 1/3，却在推理速度上大幅领先后者。此外，在准确率实验对比上，YuNet 可以在不同光照、角度、表情、遮挡等各种复杂情况实现高准确率，而后者却几乎无法进行稳定检测。显然，YuNet 已经全面优于传统的基于手工特征的人脸检测算法。值得注意的是，YuNet 和 YuNet-s 在 224×224 分辨率下几乎达到亚毫秒级推理时延，在常用的 320×320 和 640 × 480 分辨率下达到毫秒级推理时延，甚至在 1280 × 960 超大分辨率下仍能保持实时推理。

另外，为了直观展示 YuNet 在实际应用上的检测效果，本文对一张超大的自拍照进行了检测。输入分辨率为原始尺寸，置信度阈值设置为 0.5。检测结果如图 3-14所示。本文的 YuNet 能准确的检测出明显的前景人脸。剩下未检出的人脸或者太小仅有几个像素，或者太模糊难以辨认，这些人脸在实际应用中通常会被忽略。显然，YuNet 在准确率上能满足实际应用场景需求。

图 3-14　YuNet 在世界人脸最多的自拍图上的推理结果

3.5.3　消融实验

为了更深入的分析 YuNet 的性能来源，本论文进一步组织了消融实验，研究增加或删除一些组件会如何影响性能，并在表3-5中进行对比展示。

结合图3-4展示的 YuNet 网络结构细节，本文通过移除 (用符号 − 表示) 或添加 (用符号 + 表示) 一些模块来研究不同模块的功能。第一行移除最顶层的用于检测中大脸的 Stage3 和 Stage4，与其他行相比评估精度大幅下降。这表明，即使仅有很少的大脸，Stage3 和 Stage4 提取的深层特征图也是不可或缺的，它可以通过 TFPN 模块将细腻度的深层特征传递到浅层去，提高浅层的检测精度。第二行移除

表 3-5　YuNet 增减不同模块下的评估表现

网络	Stage1 通道数	Stage2,3,4 通道数	检测头堆叠数	AP_{easy}	AP_{medium}	AP_{hard}	#MFLOPs	#Params
YuNet - Top Stage3,4	64	[64]	2	0.791 (-0.101)	0.815 (-0.068)	0.744 (-0.067)	138.1	34,032
YuNet - TFPN	64	[64, 64, 64]	2	0.885 (-0.007)	0.871 (-0.012)	0.789 (-0.022)	148.7	75,856
YuNet	64	[64, 64, 64]	2	0.892	0.883	0.811	148.7	75,856
YuNet + 1x1	64	[64, 64, 64]	2	0.890 (-0.002)	0.883	0.812 (+0.001)	157.4	88,336
YuNet + Expand	64	[64, 128, 256]	2	0.906 (+0.014)	0.893 (+0.010)	0.820 (+0.009)	166.8	153,936
YuNet-s	32	[64, 64, 64]	1	0.887 (-0.005)	0.871 (-0.012)	0.768 (-0.043)	96.4	54,608

了 TFPN 模块，直观来说，移除了自顶向下的上采样操作和逐元素相加操作。与 YuNet 相比，在三个子集上精度都下降，尤其在 *Hard* 子集上下降约 2%。这说明本文提出的 TFPN 模块能实现几乎无成本的评估精度提升。第四行在 TFPN 模块中添加了用于通道数调整的 1×1 卷积。可以看到参数量大量增加的情况下，评估精度的提升却几乎忽略不计，说明该 1×1 卷积确实是用于进行通道数对齐服务的，而本文在骨干网络早已将通道数对齐到 64，那么在此处可以移除。在第五行中，本文随网络深度加深成倍增加了 Stage3 和 Stage4 的通道数量，结果显示准确率只提高了大约 1%，参数量却几乎翻倍，证明了本文提出的轻量级网络设计原则 (见章节3.2.1) 的有效性。

另一个消融实验时是关于样本尺度分布的。图3-13中展示了四种尺度缩放值集合的样本尺度分布曲线。它们的相应评估结果在表3-6中展示。显然，平均评估精度表现最好的尺度缩放集合是 [0.5, 1.5]，其分布曲线 (天蓝色) 也是最接近原始分布曲线 (红色) 的，这证明了本文提出的样本均衡的数据增强算法的有效性。

表 3-6　不同尺度缩放集合下的评估精度表现

Range	AP_{easy}	AP_{medium}	AP_{hard}	Average
[0.3, 0.8]	0.902	0.886	0.773	0.854
[0.3, 1.0]	0.901	0.885	0.794	0.860
[0.5, 1.5]*	0.892	0.883	0.811	0.862
[0.3, 2.0]	0.886	0.876	0.803	0.855

3.6　本章小结

为了解决边缘设备上人脸检测任务面临的挑战，本章首先梳理了深度学习人脸检测任务流程和主要结构，并分析领域前沿的优秀工作，总结出轻量化网络模型设计的规律。其次，基于总结的轻量化模型设计原则，手动设计了 YuNet 超轻量级人脸检测网络结构，包括骨干网络、颈部网络和检测头网络，以图表结合的

形式分析了轻量化设计的合理性。接着，为了适应轻量级人脸检测网络，使得模型在大幅降低参数量和计算量的同时，仍能保持高精度，本章制定了相应的模型训练策略，并详细阐述了提出的无锚框机制和自适应标签匹配策略。本章还分析了 WIDER-FACE 数据集中样本尺寸的分布情况，并发现改变样本尺寸分布会对模型训练造成显著影响，因此本章提出一种样本均衡数据增强策略，进一步提升了 YuNet 的评估精度。最后，本章设计了对比实验，为了公平地评估不同方法在同一测试条件下的性能差异，统一规范了测试条件，并基于开源权重实现了人脸检测流程中的前后处理环节，在统一推理引擎下对比分析了模型参数量、计算量、检测精度、推理时延等指标。另外，本章设计消融实验，对网络结构设计时各模块使用的合理性和对样本均衡的数据增强策略的有效性进行论证。

第 4 章　面向边缘设备的人脸检测库设计

4.1　项目概述

基于本文提出的 YuNet 方法，我们实现了面向边缘设备的开源 libfacedetection 轻量级人脸检测库[78]，用于快速执行图像和视频中的人脸检测任务。考虑到边缘设备的特殊性，该项目使用纯 C++ 实现，代码不依赖任何第三方库。用户可以在 Windows，Linux，ARM 和任何具有 C++ 编译器的平台下编译源代码得到人脸检测 API，可方便地集成到各种应用中。我们针对 AVX2/AVX512/NEON 指令集平台显式实现了加速代码。整个项目由训练库和运行库组成，训练库负责 YuNet 网络的训练并按规则导出序列化的权重，运行库负责人脸检测 API 的实现。下面将对两库分别进行详细介绍。

4.2　基于 MMDetection 的训练库设计

4.2.1　MMDetection 介绍及应用

MMDetection[76] 是商汤和港中文大学联合推出的一个面向目标检测任务的开源项目，它基于 Pytorch 实现了多种检测算法和数据增强技术，并支持多种数据集和模型的训练和测试。它将数据集构建、模型搭建、训练策略等过程都封装成了模块化的组件，通过组件调用的方式，使我们能够以很少的代码量实现新的检测算法，大大提高了代码复用率。MMDetection 具有灵活性、易用性、高性能和可扩展性等特点，是目前目标检测领域最流行的工具箱之一，被广泛应用于学术研究和工业实践中。

MMDetection 与 Pytorch 在框架整体流程上的差异如图4-1所示。其中蓝色部分表示 Pytorch 流程，橙色部分表示 MMDetection 流程，绿色部分表示与算法框架无关的通用流程。本文使用 MMDetection 构建 YuNet 人脸检测算法包含以下四个步骤：

(1)注册数据集：CustomDataset 是 MMDetection 在原始的 Dataset 基础上的再次封装，其 __getitem__() 方法会根据训练和测试模式分别重定向到 prepare_train_img() 和 prepare_test_img() 函数。用户以继承 CustomDataset 类的方式构建自己的数据集。本文采用 WIDER-FACE[54] 数据集，由于官方预定义的数据集实现不包含人脸关键点标注，所以我们需要重写 load_annotations() 和 get_ann_info()

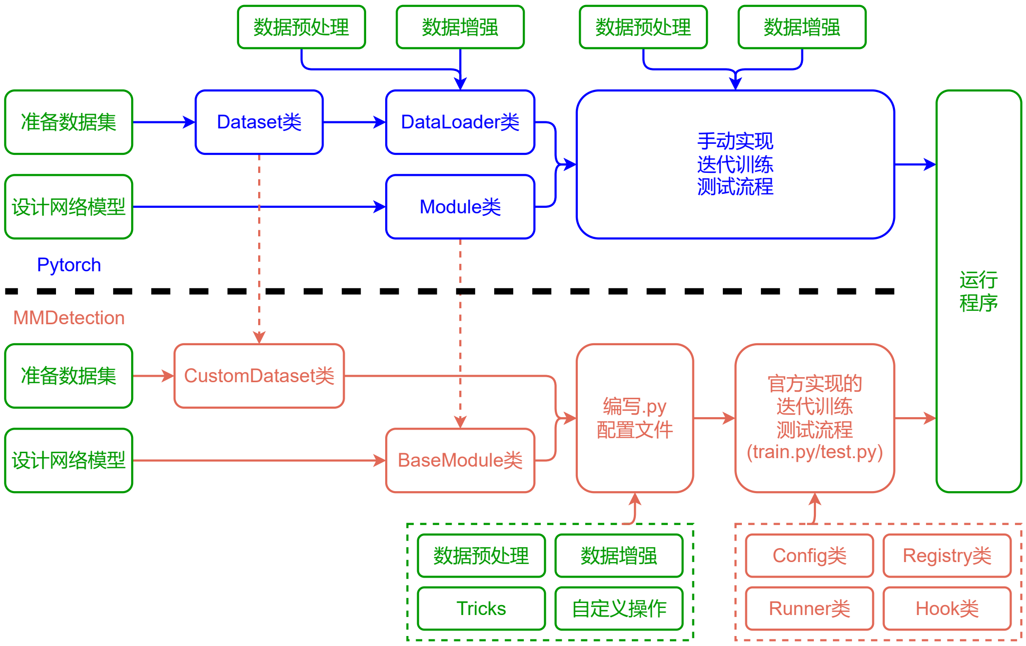


图 4-1　基于 Pytorch 和 MMDetection 的深度学习检测框架设计基本流程

函数，定义人脸外接矩形和人脸关键点数据的加载及遍历方式。完成 WIDER-FACE 数据集类的定义后，还需要使用 DATASETS.register_module() 进行模块注册。另外，本文还需要对提出的样本均衡的数据增强算法进行手动实现，参考 MMDetection 官方其他数据增强算法的输入输出，使用 PIPELINES.register_module() 进行模块注册。

(2)注册模型：模型构建的方式和 Pytorch 类似，都是新建一个 Module 的子类然后重写 forward() 函数。MMDetection 将一个完整的模型拆分为 Backbone、Neck 和 Head 三部分进行管理，本文也在章节3.2.2中将 YuNet 的构建拆分为该三部分，分别使用 BACKBONES.register_module()、NECKS.register_module() 和 HEADS.register_module() 完成模块注册。这样做能方便的控制各模块进行对比实验，例如在章节3.5的消融实验中，本文对各种模块组件快速添加或者移除。

(3)构建配置文件：配置文件用于配置检测算法的各个模块组件的组织形式和运行参数，包含四部分：datasets、models、schedules 和 runtime。完成前两个步骤定义和注册相应模块后，往配置文件中添加相应模块的运行参数。训练或测试程序执行时，MMDetection 会通过 Registry 类读取并解析配置文件，完成模块的实例化。快速修改配置文件中运行参数能明显将实验配置简单化，本文的 YuNet-s 即是在 YuNet 的基础上快速修改 Backbone 的参数得到。

(4)训练和验证：在完成各模块的代码实现、模块的注册、配置文件的编写后，仅需要对官方工具代码做少量修改就可以对模型进行训练和验证。

4.2.2　模型导出

本节主要介绍将基于 Pytorch 的.pth 权重文件导出为本文运行库支持的文件的细节。图4-2展示了权重导出的流程图。由于 YuNet 的模型参数规模非常小，我们采用最直观的方式，直接将各层权重按照“变量 ={数据}”的格式写入一个独立的 C++ 源文件中。这样做的好处是，可以在编译阶段就将模型权重写入程序，避免程序运行时的额外文件加载操作。章节3.2.2提到 YuNet 整个网络结构由基本模块 DWUnit 构建，除了首层的 ConvHead。我们可以递归遍历整个网络结构，查找 DWUnit 模块和 ConvHead 模块进行权重导出。DWUnit 模块由一个 1×1 逐点卷积、一个 3×3 逐深度卷积、批归一化层和 ReLU 激活层构成。在导出卷积层权重之前，为了减少批归一化在网络前向推理时的开销，降低实时内存空间使用并提高运算速度，我们将卷积层和批归一化层权重按照章节2.1.2的方法进行合并。合并得到的新权重替代卷积层权重进行导出，卷积权重 (包含卷积核和偏差) 数据按照 OHWI 的内存排列，保留三位有效数字。另外，ConvHead 模块的首层，输入通道数为 3 的 3×3 标准卷积将转化为输入通道数为 32 的 1×1 逐点卷积导出 (见章节4.3.2)。

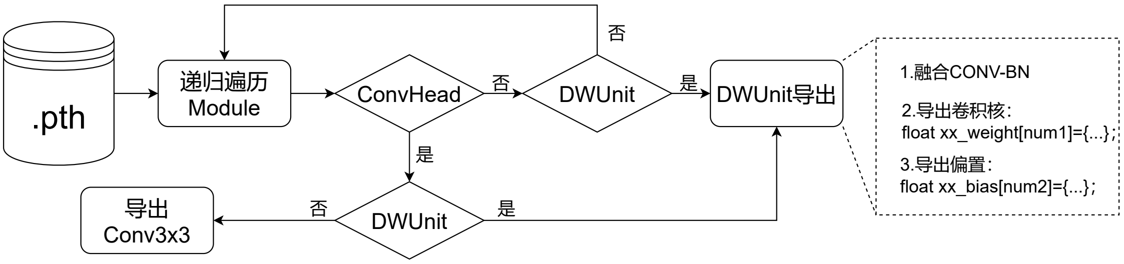


图 4-2　模型权重导出流程图

4.3　基于 SIMD 指令集的运行库设计

4.3.1　SIMD 指令集简介

SIMD（单指令多数据）是一种基于并行计算技术的计算机指令集架构。它的特点是在一条指令中，对多个数据元素执行相同的操作，从而提高计算效率。SIMD 指令集的发展始于 20 世纪 80 年代，最初主要应用于超级计算机和向量处理器。Cray-1 超级计算机上的向量指令集是最早的 SIMD 指令集，它使用了向量处理器来完成高性能的科学计算任务。随后，为了满足多媒体应用的需求，Intel 在 1996 年推出了 MMX（多媒体扩展）指令集，它可以并行处理多个数据元素，从而加速多媒体应用的运行速度。Intel 在 1999 年推出了 SSE（流式 SIMD 扩展）指令集，它在 MMX 指令集的基础上，增加了对单精度浮点数运算和更多指令的支持。Intel 在

2011 年推出了 AVX(Advanced Vector Extensions) 指令集，它是一种基于 SSE 指令集发展而来的向量指令集，它可以支持更长的数据长度，从而增强了计算能力，进一步提升处理器的性能。2013 年，Intel 在发布的 Haswell 处理器上开始支持 AVX2 指令集，它支持 256 位数据的向量操作。同年，Intel 在 Intel Xeon Phi 协处理器中引入 AVX-512 指令集，它支持 512 位数据的向量操作。除了 Intel 和 AMD 的 x86 架构之外，ARM 架构也在 ARMv7 架构中发展了自己的 SIMD 指令集。NEON 指令集与 x86 架构的 SSE 指令集类似，支持 16、32、64 位数据类型的向量操作。NEON 指令集广泛应用于移动设备、嵌入式系统和数字信号处理等领域。总之，SIMD 指令集是一种随着计算机处理器性能的提高而不断演进和完善的技术，并广泛应用于多媒体、科学计算、人工智能等领域。在未来，随着计算机处理器的继续发展，SIMD 指令集仍将发挥重要作用。

相比于普通指令，SIMD 指令在执行速度、访存和能耗方面有很多优势。SIMD 指令集能够提高处理器的性能和效率，适合大数据和高性能计算；减少指令数量和存储空间，提高执行速度；减少内存访问次数，提高内存带宽；减少处理器工作量和能耗，提高能效比和寿命。然而，SIMD 指令集也有以下缺点：适用范围有限，只能处理连续块的向量或矩阵数据，而且需要编程技巧；跨平台性差，不同的 CPU 架构和操作系统支持的 SIMD 指令集不同，编写通用的 SIMD 代码需要考虑兼容性问题；不支持条件执行，无法处理复杂的分支逻辑，可能会导致性能下降和逻辑错误；需要对数据进行内存对齐，否则可能会导致内存错误和性能下降。总之，SIMD 指令集是一种有效的并行处理技术，但也有一些限制和挑战。研究者需要根据应用场景和需求，选择合适的 SIMD 指令集和编程方法。

4.3.2　卷积计算优化方法

YuNet 的网络结构细节如图3-4所示。由于追求极致的轻量级人脸检测器，本项目选用比 YuNet 更加轻量化的 YuNet-s 网络进行实现。YuNet-s 和 YuNet 相比，Stage1 的通道数缩减为 32，且多任务解耦检测头前用于融合特征的 DWUnit 减少一层。表4-1统计了 YuNet-s 中各模块的数量和计算量开销。整个网络总共包含 6 种不同的运算模块，包括 3×3 标准卷积、1×1 标准卷积、3×3 深度卷积、ReLU、最大池化、上采样。1×1 标准卷积约占总计算量的 68%，可以近似的认为该卷积是整个网络执行的性能瓶颈。为了更好的优化内存访问，本文采用 HWC 的内存排列来组织所有特征图和卷积权重。

我们注意到 1×1 标准卷积和 3×3 深度卷积都有 26 个，而 3×3 标准卷积仅有 1 个，为方便优化，首先将 3×3 标准卷积转化为 1×1 标准卷积进行计算。转化方式如图4-3所示。输入特征图是来自预处理后的 $(H,W,3)$ 图像，图中左上角的

表 4-1　YuNet 各网络模块详细信息

模块	3×3 标准卷积	1×1 标准卷积	3×3 深度卷积	ReLU	最大池化	上采样
数量	1	26	26	14	5	2
计算量 (MFLOPs)	34.41	182.56	53.04	-	-	-
计算量占比 (%)	12.7	67.6	19.6	-	-	-

C_{in} 取值为 3，那么对图像任意 3×3 像素区域拉伸之后可以得到 $3\times9=27$ 维向量。SIMD 指令需要对内存严格对齐，所以还需要在向量尾端填充零至 32 维。然后，通过 3×3 区域在图像上的自左向右，自上而下，以 2 位步长，进行滑动扫描，得到重排后的特征图数据，重排后的维度为 $(H/2, W/2, 32)$。同样，卷积核也需要进行相似的处理，将 $(3\times3\times C_{in}\times C_{out})$ 的权重重排为 $(C_{out}, 32)$ 的向量，可以看作为输入通道为 32，输出通道为 C_{out} 的 1×1 卷积。至此，整个网络仅有 1×1 标准卷积和 3×3 深度卷积两种卷积操作，这也是本文的后续优化重点。

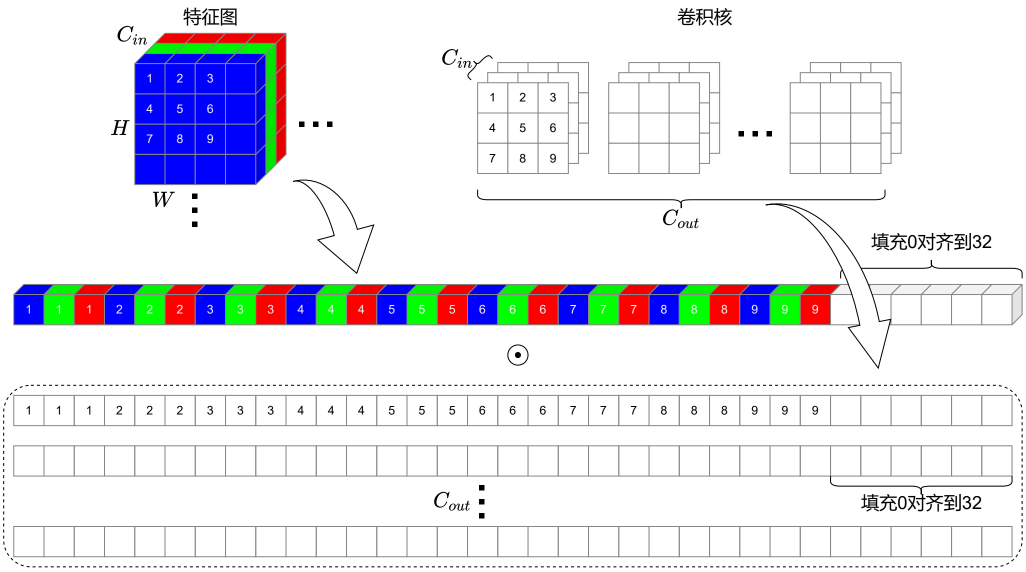
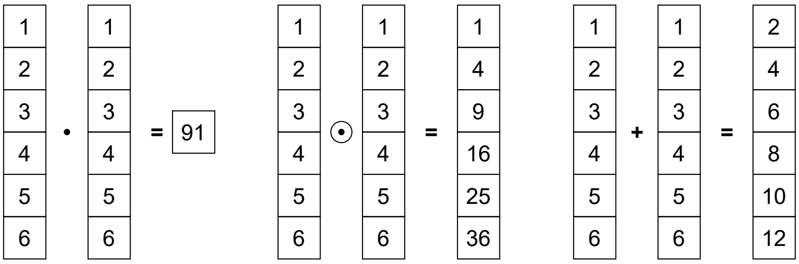


图 4-3　将 3×3 的标准卷积转化为 1×1 的标准卷积进行计算示意图

卷积计算在领域内通常被转化为已经被优化的较好的通用矩阵乘法 (General Matrix Multiplication, GEMM)。通用矩阵乘法是科学计算的核心，在各类基础线性代数库 (Basic Linear Algebra Subroutine, BLAS) 如 OpenBLAS[79] 等中已经被优化到极致，在巨大矩阵乘计算中实现比朴素矩阵计算方法快上百倍的速度提升。Chetlur 等[80] 使用 im2col 技术对输入特征图进行内存重排，将原来的卷积计算转化为通用矩阵乘法。然而，内存重排的过程会带来额外的内存访问开销，最终限制整个算法实现的性能提升。考虑到本项目是专门针对 YuNet-s 的快速实现，而 YuNet-s 的卷积核通道数都较低，最大的通道数仅为 64。如果卷积核铺开，得到的权重矩

阵维度比较小。以输入输出通道数均为 64 的 1×1 卷积核为例，所有权重仅有 $64 \times 64 = 4096$ 个 float32 字，占内存空间 16 KB。而常用处理器的 L1 Cache 总大小通常为 32 KB，因此整个卷积核权重可以直接全部装载到 L1 Cache 中。GEMM 专注于特大矩阵与特大矩阵的乘法性能提升，其性能提升的关键之一是通过 Cache 分块来降低 Cache miss 以提高计算强度[81] (Arithmetic Intensity, AI)。而小矩阵乘法本身的访存开销很小，甚至 L1 Cache 就能覆盖整个计算过程，所以 GEMM 对小矩阵乘法的性能提升较小，甚至无法抵消内存重排带来的访存开销。在 YuNet 人脸检测器的实现中，尽管输入特征矩阵的维度可以很大，但是卷积核权重矩阵太小，GEMM 的性能优化不明显反而需要进行复杂的内存重排，得不偿失。



(a)向量内积　(b)向量哈达玛积　(c)向量加

图 4-4　本文涉及的三种向量运算示意图

参考章节2.1.2对卷积计算方式的介绍，我们得知，1×1 标准卷积最基本的计算单元是两向量内积，3×3 深度卷积的最基本计算单元是两向量逐元素乘积 (在矩阵中称为向量哈达玛积) 和向量加和。三种基本计算单元的示意图如图4-4所示。本文将卷积计算优化转化为三种基本向量运算的优化。主要优化方法是将单指令单数据的朴素串行计算优化为单指令多数据的并行计算。其中涉及到的常用指令集有：AVX2，AVX512，NEON。前两者是 Intel 提出的用于 X86 架构的高级矢量计算扩展，后者是 ARM 架构的矢量计算扩展。仅支持 AVX2 指令集的 CPU，一个计算核心有 16 个 256 位寄存器，分别称为 YMM0-15，其中低 128 位复用为 XMM0-15。支持 AVX512 指令集的 CPU，一个计算核心有 32 个 512 位寄存器，分

```
float vecDot(const float* p1, const float* p2, int num){
    float sum = 0.f;
    for(int i = 0; i < num; i++){
        sum += (p1[i] * p2[i]);
    }
    return sum;
}
```

代码 4-1　朴素的向量内积计算函数。

```
float vecDot(const float* p1, const float* p2, int num){
   float sum = 0.f;
   __m256 a_float_x8, b_float_x8;
   __m256 sum_float_x8 = _mm256_setzero_ps();
   for (int i = 0; i < num; i += 8){
      a_float_x8 = _mm256_load_ps(p1 + i);
      b_float_x8 = _mm256_load_ps(p2 + i);
      sum_float_x8 = _mm256_add_ps(sum_float_x8, _mm256_mul_ps(
         a_float_x8, b_float_x8));
   }
   sum_float_x8 = _mm256_hadd_ps(sum_float_x8, sum_float_x8);
   sum_float_x8 = _mm256_hadd_ps(sum_float_x8, sum_float_x8);
   //两次调用hadd指令进行向量内部元素加和
   sum = ((float*)&sum_float_x8)[0] + ((float*)&sum_float_x8)[4];
   return sum;
}
```

代码 4-2　AVX2 指令集优化的向量内积计算函数。

别称为 ZMM0-31，其中低 256 位复用为 YMM0-31，低 128 位复用为 XMM0-31。支持 NEON 指令集的 ARMv8 处理器中，一个计算核心有 32 个 128 位寄存器，分别称为 V0-31。显然，AVX2 指令集支持一次操作 8 个 float32 数，AVX512 指令集支持一次操作 16 个 float32 数，而 NEON 指令集支持一次操作 4 个 float32 数。

由于向量内积的实现既包含向量逐元素相乘运算，也包含向量逐元素相加运算，所以本文以向量内积的优化为例，详细介绍 SIMD 指令的优化思路。向量内积，如图4-4(a) 所示，即为对相同维度的两个向量逐元素相乘，然后对所有元素求和，得到一个标量值。朴素的向量内积计算如代码4-1所示，输入是两个参与计算 float32 数组的 const 首指针和向量长度，输出是一个 float32 的向量内积值。一条代码 (第 4 行) 仅能操作 2 个 float32 值进行乘加运算。

将输入指针的起始地址对齐到 256，并保证向量长度为 8 的倍数，可以得到 AVX2 指令集下的优化计算函数如代码4-2所示。类型 __m256 表示长度为 256 位的向量，可以包含 8 个 float32 数。首先声明两个迭代变量 a_float_x8 和 b_float_x8，然后声明一个累加变量 sum_float_x8 并初始化为零。循环体步长设置为 8，每次循环首先分别将输入的各 8 个 float32 数装入两个迭代变量中，然后调用函数 _mm256_mul_ps() 执行向量按位相乘，并调用 _mm256_add_ps() 函数将结果按位累加到累加变量上。跳出循环体后，需要对累加变量进行横向求和并返回一个 float32

```
float vecDot(const float* p1, const float* p2, int num){
    float sum = 0.f;
    __m512 a_float_x16, b_float_x16;
    __m512 sum_float_x16 = _mm512_setzero_ps();
    for (int i = 0; i < num; i += 16){
        a_float_x16 = _mm512_load_ps(p1 + i);
        b_float_x16 = _mm512_load_ps(p2 + i);
        sum_float_x16 = _mm512_add_ps(sum_float_x16, _mm512_mul_ps(
            a_float_x16, b_float_x16));
    }
    //使用AVX512特有reduce指令实现向量内部加和
    sum = _mm512_reduce_add_ps(sum_float_x16);
    return sum;
}
```

代码 4-3　AVX512 指令集优化的向量内积计算函数。

值，作为向量内积的结果。AVX2 指令集没有直接进行横向求和的指令函数，本文采用两次调用 _mm256_hadd_ps() 函数再采用标量相加的方式实现。

将输入指针的起始地址对齐到 512，并保证向量长度为 16 的倍数，可以得到 AVX512 指令集下的优化计算函数，如代码4-3所示。类型 __m512 表示长度为 512 位的向量，可以包含 16 个 float32 数。首先声明两个迭代变量 a_float_x16 和 b_float_x16，然后声明一个累加变量 sum_float_x16 并初始化为零。循环体步长设置为 16，每次循环首先分别将输入的各 16 个 float32 数装入两个迭代变量中，然后调用函数 _mm512_mul_ps() 执行向量按位相乘，并调用 _mm512_add_ps() 函数将结果按位累加到累加变量上。跳出循环体后，与 AVX2 指令集不同的是，AVX512 指令集可以通过调用 _mm512_reduce_add_ps() 函数直接对向量进行横向求和并返回一个 float32 值。

将输入指针的起始地址对齐到 128 位，并保证向量的长度是 4 的倍数，可以得到如代码4-4所示的 NEON 指令集下的优化计算函数。类型 __float32x4 表示一个长度为 128 位的向量，可以包含 4 个 float32 类型的数。首先声明两个迭代变量 a_float_x4 和 b_float_x4，然后声明一个累加变量 sum_float_x4 并初始化为零。循环体的步长设置为 4，每次循环首先分别将输入的各 4 个 float32 类型的数装入两个迭代变量中，然后调用函数 vmulq_f32() 执行向量按位相乘，并调用 vaddq_f32() 函数按位累加到累加变量上。跳出循环体后，调用 vgetq_lane_f32() 函数分别访问累加变量中的 4 个 float32 类型的值并求和作为返回值。

```
float vecDot(const float* p1, const float* p2, int num){
    float sum = 0.f;
    float32x4_t a_float_x4, b_float_x4;
    float32x4_t sum_float_x4;
    sum_float_x4 = vdupq_n_f32(0);
    for (int i = 0; i < num; i+=4){
        a_float_x4 = vld1q_f32(p1 + i);
        b_float_x4 = vld1q_f32(p2 + i);
        sum_float_x4 = vaddq_f32(sum_float_x4, vmulq_f32(a_float_x4
            , b_float_x4));
    }
    //NEON指令集没有直接向量内部加和指令，使用vgetq指令进行逐元素加和
    sum += vgetq_lane_f32(sum_float_x4, 0);
    sum += vgetq_lane_f32(sum_float_x4, 1);
    sum += vgetq_lane_f32(sum_float_x4, 2);
    sum += vgetq_lane_f32(sum_float_x4, 3);
    return sum;
}
```

代码 4-4　NEON 指令集优化的向量内积计算函数。

按照与实现向量内积 vecDot() 相似的思路，本文实现了向量加和 vecAdd() 以及向量逐元素乘并加和 vecMulAdd() 等基础函数的 SIMD 优化实现。优化后的 1×1 标准卷积执行流程如代码4-5所示：

3×3 逐深度卷积和 1×1 标准卷积的优化思路一致，本文不再做详细介绍。另外本文还对 ReLU 函数、最大值池化函数、上采样函数和 NMS 函数等进行 SIMD 优化实现，由于计算量占比不大，性能提升不明显。

另外，为了实现 SIMD 优化，需要将数据地址对齐到 128bit (NEON)、256bit (AVX2) 或 512bit (AVX512)。如果读取非对齐的内存地址，可能会导致程序崩溃。为了方便地得到对齐的内存空间，本文手动实现了一个动态内存分配函数，可以满足任意数据地址对齐的要求，如图4-5所示。假设需要在堆上动态分配 s 个字节的空间，返回的数据首地址需要对齐到 128 位 (16 字节)。首先调用 malloc() 函数分配 $s+8+16$ 个字节的空间，其中 8 字节用于存储 malloc() 函数返回的首指针指向的地址 (64 位系统指针占 8 字节)，用于释放动态内存空间，16 字节是一个单位的对齐长度，用于保证对齐后有 s 个字节。假设 malloc() 函数返回的指针 (黄色) 指向内存空间首地址 0x008b(绿色)，那么本文的 myAlloc() 会将该地址存储到下一

```
void conv1x1(const CDataBlob<float> & inputData, const Filters<
    float> & filters, CDataBlob<float> & outputData){
    //三重循环实现卷积计算
    for (int row = 0; row < outputData.rows; row++){
        for (int col = 0; col < outputData.cols; col++){
            float * pOut = outputData.ptr(row, col);
            const float * pIn = inputData.ptr(row, col);
            for (int ch = 0; ch < outputData.channels; ch++){
                const float * pF = filters.weights.ptr(0, ch);
                pOut[ch] = vecDot(pIn, pF, inputData.channels);
                pOut[ch] += filters.biases.data[ch];
            }
        }
    }
}
```

代码 4-5　1 × 1 标准卷积的计算函数。

个对齐地址前的 8 个字节中 (红色)，然后返回对齐地址 0x00a0(蓝色)。当回收内存时，通过地址 0x00a0(蓝色) 往前 8 个字节找到首指针的地址 (红色) 得到该段动态空间的首地址 0x008b(绿色)，传入 free() 函数进行内存释放。

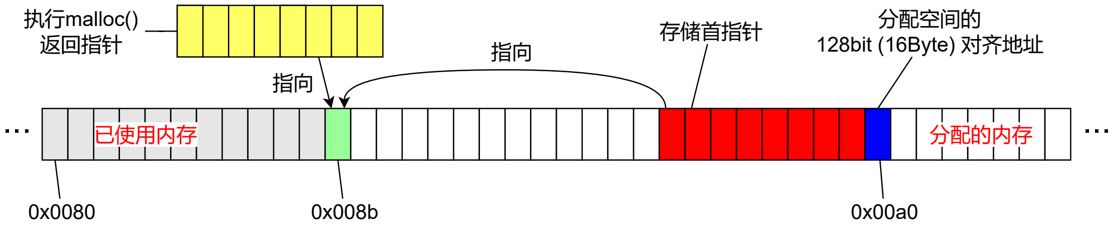


图 4-5　实现 128 位内存地址对齐的 myAlloc 函数示意图

4.3.3　总体结构

本项目的目的是实现一个适配多指令集平台 (AXV2/AVX512/NEON) 的、不依赖任何第三方库的快速人脸检测库，并编译成动态链接库供有需求的用户使用。为了方便用户使用，项目的输入为 BGR 颜色格式的图像字节流，在内存中按 HWC 排列。用户可以使用热门的 OpenCV 库以默认格式直接读取图像，或使用 PIL 库等其他图像处理库指定以 BGR 格式读取图像，得到输入字节流。人脸检测任务的执行流程如图4-6所示，总共分为五个阶段。

在检测任务开始之前，首先要进行人脸检测器的初始化。初始化操作主要是

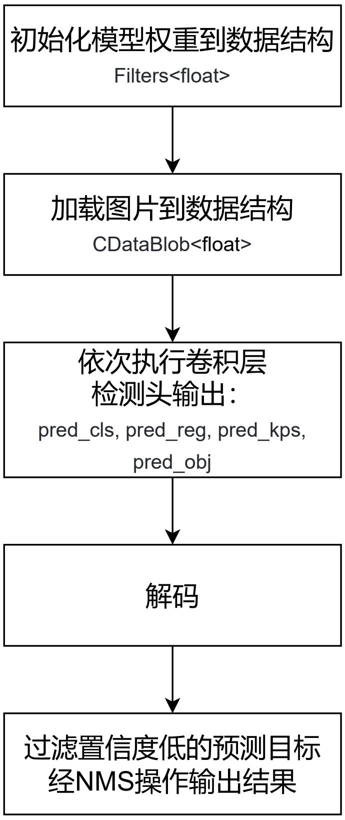


图 4-6　算法执行的总体流程

将内存中的模型权重按照一定的组织形式放入数据结构中。章节4.2.2导出的模型权重在编译阶段就直接以变量的形式写入程序中，程序启动时在栈上随机分配内存赋值，分配的内存地址通常不满足对齐要求。本文设计 Filters<float> 数据结构，重新调用 myAlloc() 函数来分配对齐的内存空间存储模型权重，使得本项目能一次初始化多次调用。

人脸检测器初始化之后，用户就可以调用人脸检测函数进行快速检测。对于任意图像输入字节流，我们先加载到 CDataBlob<float> 中。该数据结构发挥三个关键作用：填充、重排和对齐。YuNet 人脸检测器的关键输入限制是输入图片的长宽必须是 s 的倍数，其中 s 是特征图的最大步长 (见章节3.3.1)。原因如果特征图降维时长宽无法被 2 整除，会得到向下取整的新长宽值，以至于后续在特征金字塔结构中采用上采样升维操作时无法还原维度，无法执行逐元素相加的操作。本项目的最大步长为 32，因此我们将用户的输入图像的长宽进行零填充到 32 的倍数。重排作用指将输入图像按照图4-3所示的方法转换以适配 1×1 卷积代替 3×3 卷积。对齐作用即将图像数据进行内存地址对齐。

加载图像之后，按照 YuNet 的模型结构依次调用操作函数进行运算，得到 pred_cls、pred_reg、pred_kps 和 pred_obj 值，分别表示分类预测值、外接矩形框预测值、人脸关键点预测值和置信度预测值。接着将这些值按照章节3.3.1进行解

码，得到预测目标的实际映射。然后根据认为设置的置信度阈值，过滤掉置信度低的预测目标，经非极大值抑制 (NMS) 操作输出检测结果。本项目的人脸检测执行流程到此结束。

4.3.4 实验结果

本文的人脸检测器基于 YuNet-s 实现，仅有 54608 个参数，如果使用 float32 推理，模型权重仅占 213.3 KB 内存空间。测试所用硬件设备主要有两种：基于个人计算机使用 Ubuntu 操作系统，处理器是基于 X64 架构的 Intel(R) Core(TM) i7-7820X CPU@3.60GHz，具有 16 个计算核心，支持 16 线程并行计算；基于树莓派 (Raspberry PI 4B) 上使用 Centos 操作系统，处理器是 Cortex-A72 (ARMv8) 架构的 Broadcom BCM283 64-bit SoC@1.5GHz，具有 4 个计算核心，支持 4 线程并行计算。前者用于测试朴素代码 (用 NAIVE 表示)、AVX2 指令集和 AVX512 指令集下的性能表现，后者用于测试 NEON 指令集下的性能表现。本文通过控制编译选项来区分采用的指令集。由于不同的处理器因计算核心数量差距会导致多线程推理的性能变化很大，难以进行公平的性能比较，本文分别在单线程和多线程环境下进行测试，以充分比较人脸检测器的性能。本文的多线程实现基于 OpenMP，主要是用的并行模型是 OpenMP 的 Fork-join 模型，即程序的串行部分由主线程执行，遇到可以并行的代码块，主线程将任务分配给所有子线程同时执行，子任务执行完毕后，主线程进行线程回收并继续执行后续的串行部分。本文的人脸检测主函数是单线程实现，当测试多线程情况下的表现时，在测试代码中以预编译命令 #pragma omp 来告诉编译器需要并行处理的代码块，即循环调用人脸检测主函数的部分。

本文通过测试在不同输入分辨率下的推理时间和帧率来比较性能。我们先循环执行人脸检测主函数 10 个轮次且不计入总时间，以排除检测器初始化时间开销和内存加载的干扰，然后循环执行 256 个轮次，计算每轮执行的平均延迟来确定推理时间。对推理时间取倒数得到帧率。测试结果如表4-2所示。对比 AVX2/AVX512 指令集和 NAIVE 情况下的推理时延，可以看出我们的 SIMD 实现在各分辨率下都有约 3 倍的速度胜出。应用 AVX2/AVX512，我们在各分辨率下都实现了多线程毫秒级推理。在 NEON 指令集下，我们在输入分辨率为 320×240 时通过多线程方式实现了实时推理。显然，我们提出的基于 YuNet-s 的人脸检测器在不依赖任何第三方库 (如线性代数库或者专用推理引擎) 的情况下实现了非常好的性能，当部署到边缘设备时，占用存储和内存空间少 (约 200 KB)，推理速度快 (毫秒级)，检测精度高 (大幅优于基于传统机器学习算法)，完美符合边缘设备的使用需求。

表 4-2　各指令集下推理速度表现

指令集	处理器	分辨率	推理时间 (ms)	帧率	推理时间 (ms)	帧率
			单线程	单线程	多线程	多线程
NAIVE	Intel(R) Core(TM)	640 × 480	157.7	6.3	15.1	66.4
	i7-7820X CPU@3.60GHz	320 × 240	40.7	24.5	4.1	245.5
	16 核 16 线程	160 × 120	10.7	93.6	1.2	802.7
	X64	128 × 96	6.5	153.3	0.7	1,478.9
AVX2	Intel(R) Core(TM)	640 × 480	50.02	20.0	6.55	152.7
	i7-7820X CPU@3.60GHz	320 × 240	13.09	76.4	1.82	550.5
	16 核 16 线程	160 × 120	3.61	277.4	0.57	1,745.1
	X64	128 × 96	2.11	476.6	0.33	2,994.2
AVX512	Intel(R) Core(TM),	640 × 480	46.47	21.5	6.39	156.5
	i7-7820X CPU@3.60GHz	320 × 240	12.10	82.7	1.67	599.3
	16 核 16 线程	160 × 120	3.37	296.5	0.46	2,155.8
	X64	128 × 96	1.98	504.72	0.31	3,198.6
NEON	Raspberry Pi 4B (Linux)	640 × 480	404.63	2.5	125.47	8.0
	Broadcom BCM2835	320 × 240	105.73	9.5	32.98	30.3
	Cortex-A72(ARMv8) 64-bit SoC@1.5GHz	160 × 120	20.05	38.4	7.91	126.5
	4 核 4 线程	128 × 96	15.06	66.4	4.5	222.3

4.4　本章小结

本章的主要内容是介绍了如何在边缘设备上部署和优化 YuNet 人脸检测网络，并开发了 libfacedetection 开源项目。为了实现这一目标，本章首先基于 MMDetection 开源库搭建了 YuNet 模型的训练程序，详细说明了数据集的注册、模型的定义、配置文件的编写以及训练和验证的过程。然后为了方便在 C++ 环境下加载模型权重，本章采用了一种简单有效的导出方式，即将模型权重转换成“变量 = 数据”的形式，并保存到 C++ 源文件中。然后，本章简述了 SIMD 指令集的发展，总结了应用 SIMD 指令集来优化矩阵计算的关键。接着为了提高在边缘设备上的运行速度，本章针对多个指令集平台，对卷积运算中常用的基本算子进行了显式 SIMD 优化，包括向量乘加算子、向量和算子等，并结合代码详细介绍了优化方法和技巧。最后本章在多种操作系统和指令集平台下，模拟边缘设备环境对实现的人脸检测程序设计实验，进行速度测试并分析，验证了 YuNet 在边缘设备上具有显著优势。

第 5 章　结　论

5.1　论文总结

本文研究了一种轻量级人脸检测方法，以在低功耗、低性能、低存储的边缘设备或嵌入式设备上快速准确地识别图像中的人脸区域。轻量级人脸检测技术对这类设备有以下优势：提高用户体验，实现实时人脸检测，支持自动对焦、拍照、美颜等功能；降低成本，减少对云端服务器的依赖，节省网络带宽和存储空间，降低运维成本和安全风险；扩展应用场景，支持智能门禁、刷脸支付、人证比对等服务，提升服务质量和安全性。轻量级人脸检测技术也面临着以下挑战：如何在模型小巧的情况下，提高模型的精度和鲁棒性，适应不同光照、角度、表情、遮挡等复杂情况下的人脸检测；如何在模型快速的情况下，减少模型的计算量和功耗，适应不同性能和电池容量等硬件条件下的人脸检测；如何在模型通用的情况下，优化模型的架构和参数，适应不同分辨率和尺度等输入条件下的人脸检测。

基于上述背景，本论文提出了 YuNet 超轻量级人脸检测算法，它能够在保持高精度的同时，大幅减少模型参数量和计算量，特别是在参数量上实现了数量级的降低。YuNet 算法不仅全面优于传统的基于手工设计特征的人脸检测算法，而且在具有竞争力的检测精度下，实现了毫秒级的检测速度。本论文还针对边缘设备开发了 libfacedetection 开源项目，使用 C++ 实现且不依赖任何第三方库，在多个指令集平台上进行了代码层面的 SIMD 指令优化加速，并实现了快速部署。

本论文的主要内容总结如下：

(1) 通过查阅相关文献，分析了基于手工特征与基于深度学习两类人脸检测算法。发现前者虽然速度快、计算存储开销小，但是精度较低，在复杂环境下难以满足人脸检测任务需求；而后者虽然精度高、鲁棒性强，在复杂环境下能够有效地检测人脸区域，但是计算存储资源消耗大，在边缘设备上部署困难。此外，在已有轻量级人脸检测算法中也存在精度不足或速度慢等问题。

(2) 对人脸检测任务流程及主要结构进行梳理，提出了轻量化人脸检测模型的设计原则，包括：通道数不必随网络深度成倍增加，尽可能降低模型层数，减少大尺寸特征图上的卷积运算，降低网络的碎片化程度。基于这些原则，本文设计了 YuNet 超轻量级人脸检测网络，并从主干网络、颈部网络和检测头网络三个方面详细阐述了其设计合理性和实现细节。为了适应轻量级人脸检测网络，本文还制定了相应的模型训练策略，并创新性地提出了无锚框机制和自适应标签匹配策

略，使得模型在大幅降低参数量和计算量的同时，仍能保持高精度。此外，本文还分析了 WIDER-FACE 数据集中样本尺寸的分布情况，并发现改变样本尺寸分布会对模型训练造成显著影响。因此，本文提出了一种样本均衡数据增强策略，进一步提升了 YuNet 的评估精度。最后，在进行对比实验时，本文注意到很多同领域工作存在测试条件不一致的问题，在测试速度和精度时不同方法之间有明显差异。为了更加公平地进行实验，本文对测试条件进行统一规范，并基于开源权重对人脸检测流程中的前后处理等环节进行统一实现，在统一推理引擎 (ONNXRuntime) 下对不同输入分辨率下的模型参数量、计算量、检测精度、推理时延进行定量对比分析。

(3) 针对边缘设备实现了基于 YuNet 人脸检测网络的 libfacedetection 开源项目。首先介绍了模型训练程序搭建过程，即基于 MMDetection 开源库，注册数据集、注册模型、构建配置文件以及进行训练和验证等步骤，得到 YuNet 模型权重。然后由于权重文件只有 200 多 KB，为了最大化 IO 传输速度，将模型权重以“变量 = 数据”的形式导出到 C++ 源文件中，在编译阶段加载，避免运行时加载权重文件产生的额外开销。接着详细介绍了在多个指令集平台下的几种卷积运算中涉及的基本算子的显式 SIMD 优化方法，包括向量乘加算子、向量和算子等。最后在多种操作系统和指令集平台下，模拟边缘设备环境对实现的人脸检测程序进行速度测试并分析。

5.2　未来展望

本论文根据资源受限的边缘设备的低功耗、低性能和低存储特点设计了超轻量级的具有毫秒级推理速度的人脸检测网络 YuNet，并在 libfacedetection 项目中成功进行部署实现。但是仍存在值得进一步研究之处：

(1) 本论文中的 YuNet 网络结构是基于人工经验设计的，可能存在一些结构上的不必要。未来可以利用神经架构搜索 (Neural Architecture Search, NAS) 技术，根据本论文总结的轻量级人脸检测网络设计原则，自动寻找更优化的网络结构。

(2) 本论文开发的 libfacedetection 库，虽然在运行速度上做了深入优化，但是只能适配 YuNet 算法，扩展性不强。后续可以对基础内核算子进行更通用的优化，以支持更多形状和尺寸的卷积运算。

(3) 在实际边缘设备场景中，除了人脸检测外，还有其他诸如车牌检测、手势检测识别等任务需求。后续可以借鉴人脸检测的经验，将超轻量级检测方法应用到这些任务中去，打造一个全面的超轻量级检测工具箱，提升实际应用价值。

参考文献

[1] YANG M H, KRIEGMAN D, AHUJA N. Detecting faces in images: a survey[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(1): 34-58. DOI: 10.1109/34.982883.

[2] YANG G, HUANG T S. Human face detection in a complex background[J/OL]. Pattern Recognition, 1994, 27(1): 53-63. DOI: 10.1016/0031-3203(94)90017-5.

[3] LEUNG T, BURL M, PERONA P. Finding faces in cluttered scenes using random labeled graph matching[C/OL]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 1995: 637-644. DOI: 10.1109/iccv.1995.466878.

[4] YOW K C, CIPOLLA R. Feature-based human face detection[J/OL]. Image and Vision Computing, 1997, 15(9): 713-735. DOI: 10.1016/S0262-8856(97)00003-6.

[5] MCKENNA S J, GONG S, RAJA Y. Modelling facial colour and identity with gaussian mixtures [J/OL]. Pattern Recognition, 1998, 31(12): 1883-1892. DOI: 10.1016/S0031-3203(98)00066-1.

[6] KJELDSEN R, KENDER J. Finding skin in color images[C/OL]//Proceedings of the Second International Conference on Automatic Face and Gesture Recognition. 1996: 312-317. DOI: 10.1109/AFGR.1996.557283.

[7] CRAW I, TOCK D, BENNETT A. Finding face features[C/OL]//Computer Vision — ECCV’92. Springer Berlin Heidelberg, 1992: 92-96. DOI: 10.1007/3-540-55426-2_12.

[8] LANITIS A, TAYLOR C, COOTES T. Automatic face identification system using flexible appearance models[J/OL]. Image and Vision Computing, 1995, 13(5): 393-401. DOI: 10.1016/0262-8856(95)99726-H.

[9] TURK M, PENTLAND A. Eigenfaces for recognition[J/OL]. Journal of Cognitive Neuroscience, 1991, 3(1): 71-86. DOI: 10.1162/jocn.1991.3.1.71.

[10] SUNG K K, POGGIO T. Example-based learning for view-based human face detection[J/OL]. IEEE Transactions on pattern analysis and machine intelligence, 1998, 20(1): 39-51. DOI: 10.1109/34.655648.

[11] ROWLEY H, BALUJA S, KANADE T. Neural network-based face detection[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998, 20(1): 23-38. DOI: 10.1109/34.655647.

[12] OSUNA E, FREUND R, GIROSIT F. Training support vector machines: an application to face detection[C/OL]//Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1997: 130-136. DOI: 10.1109/cvpr.1997.609310.

[13] VIOLA P A, JONES M J. Rapid object detection using a boosted cascade of simple features [C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2001: 511-518. DOI: 10.1109/cvpr.2001.990517.

[14] CROW F C. Summed-area tables for texture mapping[J/OL]. SIGGRAPH Comput. Graph., 1984, 18(3): 207–212. DOI: 10.1145/964965.808600.

[15] FREUND Y, SCHAPIRE R E. A decision-theoretic generalization of on-line learning and an application to boosting[J/OL]. Journal of Computer and System Sciences, 1997, 55(1): 119-139. DOI: 10.1006/jcss.1997.1504.

[16] FELZENSZWALB P F, GIRSHICK R B, MCALLESTER D, et al. Object detection with discriminatively trained part-based models[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010, 32(9): 1627-1645. DOI: 10.1109/TPAMI.2009.167.

[17] FELZENSZWALB P F, HUTTENLOCHER D P. Pictorial structures for object recognition [J/OL]. International journal of computer vision, 2005, 61: 55-79. DOI: 10.1023/B:VISI.0000042934.15159.49.

[18] FENG Y, YU S, PENG H, et al. Detect faces efficiently: a survey and evaluations[J/OL]. IEEE Transactions on Biometrics, Behavior, and Identity Science, 2022, 4(1): 1-18. DOI: 10.1109/TBIOM.2021.3120412.

[19] REN S, HE K, GIRSHICK R B, et al. Faster R-CNN: towards real-time object detection with region proposal networks[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, 39(6): 1137-1149. DOI: 10.1109/TPAMI.2016.2577031.

[20] LI H, LIN Z, SHEN X, et al. A convolutional neural network cascade for face detection[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2015: 5325-5334. DOI: 10.1109/cvpr.2015.7299170.

[21] ZHANG K, ZHANG Z, LI Z, et al. Joint face detection and alignment using multitask cascaded convolutional networks[J/OL]. IEEE Signal Processing Letters, 2016, 23(10): 1499-1503. DOI: 10.1109/LSP.2016.2603342.

[22] WANG H, LI Z, JI X, et al. Face r-cnn[M/OL]. arXiv, 2017. DOI: 10.48550/arXiv.1706.01061.

[23] HU P, RAMANAN D. Finding tiny faces[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2017: 951-959. DOI: 10.1109/cvpr.2017.166.

[24] NAJIBI M, SAMANGOUEI P, CHELLAPPA R, et al. Ssh: single stage headless face detector [C/OL]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2017: 4875-4884. DOI: 10.1109/iccv.2017.522.

[25] ZHANG S, ZHU X, LEI Z, et al. S3fd: single shot scale-invariant face detector[C/OL]//Proceedings of the IEEE International Conference on Computer Vision. 2017: 192-201. DOI: 10.1109/iccv.2017.30.

[26] TANG X, DU D K, HE Z, et al. Pyramidbox: a context-assisted single shot face detector [C/OL]//Proceedings of the European Conference on Computer Vision. 2018: 797-813. DOI: 10.1007/978-3-030-01240-3_49.

[27] LI J, WANG Y, WANG C, et al. Dsfd: dual shot face detector[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 5060-5069. DOI: 10.1109/cvpr.2019.00520.

[28] DENG J, GUO J, VERVERAS E, et al. Retinaface: single-shot multi-level face localisation in the wild[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020. DOI: 10.1109/cvpr42600.2020.00525.

[29] LIU Y, TANG X, HAN J, et al. Hambox: delving into mining high-quality anchors on face detection[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, 2020: 13043-13051. DOI: 10.1109/cvpr42600.2020.01306.

[30] LIU Y, TANG X. Bfbox: searching face-appropriate backbone and feature pyramid network for face detector[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 13568-13577. DOI: 10.1109/cvpr42600.2020.01358.

[31] LI J, ZHANG B, WANG Y, et al. Asfd: automatic and scalable face detector[C/OL]// Proceedings of the ACM International Conference on Multimedia. Association for Computing Machinery, 2021: 2139–2147. DOI: 10.1145/3474085.3475372.

[32] ZHANG S, ZHU X, LEI Z, et al. Faceboxes: a cpu real-time face detector with high accuracy [C/OL]//2017 IEEE International Joint Conference on Biometrics (IJCB). 2017: 1-9. DOI: 10.1109/BTAS.2017.8272675.

[33] HE Y, XU D, WU L, et al. LFFD: a light and fast face detector for edge devices[M/OL]. arXiv, 2019. DOI: 10.48550/arXiv.1904.10633.

[34] QI D, TAN W, YAO Q, et al. Yolo5face: why reinventing a face detector[C]//Proceedings of the European Conference on Computer Vision, Workshops. Springer Nature Switzerland, 2023: 228-244.

[35] JOCHER G. Yolov5[J/OL]. GitHub repository, 2020. https://github.com/ultralytics/yolov5.

[36] GUO J, DENG J, LATTAS A, et al. Sample and computation redistribution for efficient face detection[C]//International Conference on Learning Representations. OpenReview.net, 2022.

[37] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J/OL]. Proceedings of the IEEE, 1998, 86(11): 2278-2324. DOI: 10.1109/5.726791.

[38] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]//Advances in Neural Information Processing Systems. 2012: 1106-1114.

[39] DENG J, DONG W, SOCHER R, et al. Imagenet: a large-scale hierarchical image database [C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2009: 248-255. DOI: 10.1109/cvpr.2009.5206848.

[40] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[M/OL]. arXiv, 2014. DOI: 10.48550/arXiv.1409.1556.

[41] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C/OL]// Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2016. DOI: 10.1109/cvpr.2016.90.

[42] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications[M/OL]. arXiv, 2017. DOI: 10.48550/arXiv.1704.04861.

[43] SANDLER M, HOWARD A, ZHU M, et al. Mobilenetv2: inverted residuals and linear bottlenecks[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018: 4510-4520. DOI: 10.1109/cvpr.2018.00474.

[44] HOWARD A, SANDLER M, CHU G, et al. Searching for mobilenetv3[C/OL]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 1314-1324. DOI: 10.1109/iccv.2019.00140.

[45] CHOLLET F. Xception: deep learning with depthwise separable convolutions[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2017: 1251-1258. DOI: 10.1109/cvpr.2017.195.

[46] XU B, WANG N, CHEN T, et al. Empirical evaluation of rectified activations in convolutional network[M/OL]. arXiv, 2015. DOI: 10.48550/arXiv.1505.00853.

[47] JARRETT K, KAVUKCUOGLU K, RANZATO M, et al. What is the best multi-stage architecture for object recognition?[C/OL]//2009 IEEE 12th International Conference on Computer Vision. 2009: 2146-2153. DOI: 10.1109/iccv.2009.5459469.

[48] NAIR V, HINTON G E. Rectified linear units improve restricted boltzmann machines[C]//Proceedings of the International Conference on International Conference on Machine Learning. Omnipress, 2010: 807–814.

[49] IOFFE S, SZEGEDY C. Batch normalization: accelerating deep network training by reducing internal covariate shift[C]//Proceedings of the International Conference on Machine Learning: volume 37. JMLR.org, 2015: 448-456.

[50] GIRSHICK R B. Fast R-CNN[C/OL]//Proceedings of the IEEE International Conference on Computer Vision. IEEE Computer Society, 2015: 1440-1448. DOI: 10.1109/iccv.2015.169.

[51] YU J, JIANG Y, WANG Z, et al. Unitbox: an advanced object detection network[C/OL]//Proceedings of the ACM International Conference on Multimedia. Association for Computing Machinery, 2016: 516–520. DOI: 10.1145/2964284.2967274.

[52] PENG H, YU S. A systematic iou-related method: beyond simplified regression for better localization[J/OL]. IEEE Transactions on Image Processing, 2021, 30: 5032-5044. DOI: 10.1109/TIP.2021.3077144.

[53] JAIN V, LEARNED-MILLER E. Fddb: a benchmark for face detection in unconstrained settings: UM-CS-2010-009[R]. University of Massachusetts, Amherst, 2010.

[54] YANG S, LUO P, LOY C C, et al. WIDER FACE: A face detection benchmark[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2016: 5525-5533. DOI: 10.1109/cvpr.2016.596.

[55] LIU Y, WANG F, DENG J, et al. Mogface: towards a deeper appreciation on face detection [C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 4093-4102. DOI: 10.1109/cvpr52688.2022.00406.

[56] ZHU Y, CAI H, ZHANG S, et al. Tinaface: Strong but simple baseline for face detection[M/OL]. arXiv, 2020. DOI: 10.48550/arXiv.2011.13183.

[57] ZHU C, ZHENG Y, LUU K, et al. Cms-rcnn: contextual multi-scale region-based cnn for unconstrained face detection[J/OL]. Deep learning for biometrics, 2017: 57-79. DOI: 10.1007/978-3-319-61657-5_3.

[58] WANG Y, JI X, ZHOU Z, et al. Detecting faces using region-based fully convolutional networks [M/OL]. arXiv, 2017. DOI: 10.48550/arXiv.1709.05256.

[59] LIN T Y, DOLLAR P, GIRSHICK R, et al. Feature pyramid networks for object detection [C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2017. DOI: 10.1109/cvpr.2017.106.

[60] LIU S, QI L, QIN H, et al. Path aggregation network for instance segmentation[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018: 8759-8768. DOI: 10.1109/cvpr.2018.00913.

[61] GHIASI G, LIN T Y, LE Q V. Nas-fpn: learning scalable feature pyramid architecture for object detection[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 7036-7045. DOI: 10.1109/cvpr.2019.00720.

[62] GE Z, LIU S, WANG F, et al. Yolox: exceeding yolo series in 2021[M/OL]. arXiv, 2021. DOI: 10.48550/arXiv.2107.08430.

[63] TIAN Z, SHEN C, CHEN H, et al. FCOS: fully convolutional one-stage object detection [C/OL]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 9627-9636. DOI: 10.1109/iccv.2019.00972.

[64] ZHANG S, CHI C, YAO Y, et al. Bridging the gap between anchor-based and anchor-Free detection via adaptive training sample selection[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Computer Vision Foundation / IEEE, 2020: 9756-9765. DOI: 10.1109/cvpr42600.2020.00978.

[65] MA N, ZHANG X, ZHENG H T, et al. Shufflenet v2: practical guidelines for efficient cnn architecture design[C/OL]//Proceedings of the European Conference on Computer Vision. Springer, 2018. DOI: 10.1007/978-3-030-01264-9_8.

[66] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. Squeezenet: alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size[M/OL]. arXiv, 2016. DOI: 10.48550/arXiv.1602.07360.

[67] REDMON J, FARHADI A. Yolov3: an incremental improvement[M/OL]. arXiv, 2018. DOI: 10.48550/arXiv.1804.02767.

[68] LIU W, ANGUELOV D, ERHAN D, et al. SSD: single shot multibox detector[C/OL]//Proceedings of the European Conference on Computer Vision. Springer, 2016: 21-37. DOI: 10.1007/978-3-319-46448-0_2.

[69] ZHOU X, WANG D, KRäHENBüHL P. Objects as points[M/OL]. arXiv, 2019. DOI: 10.48550/arXiv.1904.07850.

[70] LAW H, DENG J. Cornernet: detecting objects as paired keypoints[C/OL]//Proceedings of the European Conference on Computer Vision. 2018. DOI: 10.1007/978-3-030-01264-9_45.

[71] SHRIVASTAVA A, GUPTA A, GIRSHICK R. Training region-based object detectors with online hard example mining[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2016. DOI: 10.1109/cvpr.2016.89.

[72] GE Z, LIU S, LI Z, et al. Ota: optimal transport assignment for object detection[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Computer Vision Foundation / IEEE, 2021: 303-312. DOI: 10.1109/cvpr46437.2021.00037.

[73] ZHANG X, WAN F, LIU C, et al. Freeanchor: learning to match anchors for visual object detection[C]//Advances in Neural Information Processing Systems. 2019: 147-155.

[74] CUBUK E D, ZOPH B, MANE D, et al. Autoaugment: learning augmentation strategies from data[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019. DOI: 10.1109/CVPR.2019.00020.

[75] BOCHKOVSKIY A, WANG C Y, LIAO H Y M. Yolov4: optimal speed and accuracy of object detection[M/OL]. arXiv, 2020. DOI: 10.48550/arXiv.2004.10934.

[76] CHEN K, WANG J, PANG J, et al. MMDetection: open mmlab detection toolbox and benchmark[M/OL]. arXiv, 2019. DOI: 10.48550/arXiv.1906.07155.

[77] BAZAREVSKY V, KARTYNNIK Y, VAKUNOV A, et al. Blazeface: sub-millisecond neural face detection on mobile gpus[M/OL]. arXiv, 2019. DOI: 10.48550/arXiv.1907.05047.

[78] YU S, WU W, FENG Y. Libfacedetection[J/OL]. GitHub repository, 2020. https://github.com/ShiqiYu/libfacedetection,https://github.com/ShiqiYu/libfacedetection.train.

[79] ZHANG X. Openblas[J/OL]. GitHub repository, 2012. https://github.com/xianyi/OpenBLAS.

[80] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. Cudnn: efficient primitives for deep learning[M/OL]. arXiv, 2014. DOI: 10.48550/arXiv.1410.0759.

[81] GOTO K, GEIJN R A V D. Anatomy of high-performance matrix multiplication[J/OL]. ACM Transactions on Mathematical Software, 2008, 34(3). DOI: 10.1145/1356052.1356053.

致　谢

在这篇学位论文的最后，我要向所有帮助过我的人表达我最诚挚的感谢。没有你们的支持和鼓励，我不可能完成这样一项艰巨而有意义的任务。你们是我学习和生活中的贵人，你们是我心中的明灯，你们是我永远的朋友。

首先，我要衷心地感激我的导师——于仕琪教授。他在我的毕业论文从选题到定稿的整个过程中给予了我无微不至的指导、耐心地修改了我的论文，并提出了宝贵的意见和建议。他不仅教会了我专业知识、科研方法、写作技巧等方面的技能，更让我受益于他为人处世、做事态度、追求卓越等方面的品格。他是我的良师益友，也是我敬仰与学习的榜样。

其次，我要向 347b 实验室的同窗们表示衷心地感激，在这三年里，我们一起奋斗、一起进步、一起成长。他们在忙碌的研究生活中给与了我鼓励和支持，在遇到困难时给与了我帮助和建议，在闲暇之余给与了我友谊和欢乐。他们让我的研究生活充满了色彩和动力。

再次，对我南科大朋友们尤其是我室友表示感激，在这三年里，我们相处得非常融洽和愉快，一起奋斗的感觉真的很好。感激室友晚上回宿舍经常给我带小食品加餐，使我免去睡前的饥肠辘辘。你们是我的好兄弟，也是我的好伙伴。

再次，我要向我的父母表达深深地爱意和诚挚地感谢。在我攻读硕士学位期间，你们始终给予了我无限量的理解、支持和鼓励。你们是我的坚强后盾，在遇到困难时给予我力量，在遇到挫折时给予我安慰，在遇到成功时给予我祝福。没有你们就没有今天的我。

最后，但并非最不重要地，我要向我的妻子以及刚刚两岁的儿子说声：对不起！也说声：谢谢！对不起这三年来不能一直陪伴在你们身边；对不起不能为家庭分担更多责任；对不起不能陪你们玩玩具和收拾小细娃的屎尿屁。

研究生三年，也是疫情三年，向默默付出的各行各业工作者致敬，我们共同面对并战胜了无数困难与挑战，之后的日子里仍需共勉。百年未有之大变局，我来了！

个人简历、在学期间完成的相关学术成果

个人简历

1995 年 04 月 14 日出生于重庆。

2013 年 09 月考入重庆大学土木工程学院建筑工程专业，同期于重庆大学计算机学院修习计算机科学与技术第二专业，2017 年 06 月本科毕业并获得工学学士学位。

2020 年 09 月至今，在南方科技大学计算机科学与工程系电子科学与技术专业攻读硕士学位。

获奖情况：优秀学生助理。

工作经历：2017 年 07 月——2019 年 07 月中交广州航道局有限公司，任驻船工程师。

在学期间完成的相关学术成果

学术论文

[1] **WU W**, PENG H, YU S. YuNet: a tiny millisecond-level face detector[J]. Machine Intelligence Research, 2023: 1-10. DOI: 10.1007/s11633-023-1423-y. (EI 检索，IF=1.976，对应学位论文第 3 章.)

[2] SHEN C, FAN C, **WU W**, et al. LidarGait: benchmarking 3D gait recognition with point clouds[C/OL]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023. https://openreview.net/forum?id=M5SQiuNPD46. (已接收未发表, EI 检索, 涉及学位论文第 4 章)