软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005938]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向边缘计算的目标追踪应用部署策略*

张 展, 张宪琦, 左德承, 付国栋

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150000)

通讯作者: 张展, E-mail: zhangzhan@hit.edu.cn

摘 要: 目标追踪算法已在诸多领域得到广泛应用,然而由于实时性和功耗问题,使得基于深度学习模型的算法难以在移动终端设备上部署应用.本文结合边缘计算技术,从应用部署优化的角度,对目标追踪算法在移动设备上的部署策略进行研究.通过对目标追踪应用特点、移动设备特性以及边缘云网络架构的分析,提出一种面向边缘计算的目标追踪应用部署策略.通过任务分割策略将目标追踪应用的计算任务合理卸载至边缘云并利用信息融合策略对计算结果进行分析融合,此外,利用运动检测进一步降低终端节点的计算压力和功耗.通过对不同部署策略进行对比实验,实验结果表明,相比计算任务本地计算,该部署策略明显降低了任务响应时间,相比完全卸载至边缘云,该部署策略降低了相同计算任务的处理时间.

关键词: 目标追踪;边缘计算;资源分配;深度学习;移动计算

中图法分类号: TP311

中文引用格式: 张展,张宪琦,左德承,付国栋.面向边缘计算的目标追踪应用部署策略研究.软件学报,2020,31(9)

英文引用格式: Zhang Z, Zhang XQ, Zuo DC, Fu GD. Research on Target Tracking Application Deployment Strategy for Edge Computing. Ruan Jian Xue Bao/Journal of Software, 2020,31(9) (in Chinese).

Target Tracking Application Deployment Strategy for Edge Computing

ZHANG Zhan, ZHANG Xian-Qi, ZUO De-Cheng, FU Guo-Dong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150000, China)

Abstract: Target tracking algorithm has been widely used in many fields. However, due to the problems of real-time and power consumption, it is difficult to deploy the algorithm based on deep learning model on mobile terminal devices. This paper studies the deployment strategy of target tracking algorithm on mobile devices from the perspective of application deployment optimization combined with edge computing technology. A deployment strategy of target tracking application oriented to edge computing is proposed based on the analysis of device characteristics and edge cloud network architecture,. The computing task of target tracking application is reasonably unloaded to edge cloud by task segmentation strategy and the computing results are analyzed and fused by the information fusion strategy. In addition, a motion detection scheme is proposed to further reduce the computing pressure and power consumption of terminal nodes The experimental results show that compared with local computing, the deployment strategy significantly reduces the response time of the task, and compared with completely uninstalling to the edge cloud, the deployment strategy reduces the processing time of the same computing task.

Key words: target tracking; edge computing; target tracking; resource allocation; deep learning; mobile computing

目标追踪算法近年来在很多领域得到广泛应用,其主要应用场景包括自动驾驶、安防监控、异常行为分析等方面.例如,机器人通过目标检测进行行人识别后,通过追踪算法保持用户跟随,从而进行定向服务;工业加工

* 基金项目: 国家高技术研究发展计划(863 计划)(2013AA01A215)
Foundation item: National High Technology Research and Development Program of China (863 Program) (2013AA01A215)
收稿时间: 2019-06-27; 修改时间: 2019-08-18; 采用时间: 2019-11-02; jos 在线出版时间: 2020-01-13
CNKI 网络优先出版: 2020-01-14 11:26:54, http://kns.cnki.net/kcms/detail/11.2560.TP.20200114.1126.020.html

中,通过调整舵机保持追踪物体处于画面中心,从而使设备追踪目标进行加工等.追踪算法的主要挑战包括目标遮挡,尺度更改,光照变化以及运动模糊等多个方面.目前,主要的追踪算法可以大体分为两类,一类基于相关滤波器(CF, Correlation Filter),另一类基于深度学习(DL,Deep Learning).采用相关滤波器的算法,将传入的原始数据经过快速傅里叶变换转换到频域内进行计算,大大提高了计算速度,适用于计算能力较弱的移动设备.基于深度学习的追踪算法在近年来成为一大热点.该类方法的主要缺点在于计算压力较大,虽然可以结合模型压缩技术对深度学习模型进行压缩,但仍难以达到追踪任务对于算法实时性的严苛要求,尤其对于计算能力较为薄弱的嵌入式设备,难以部署应用.

嵌入式设备较弱的计算能力一直是其应用瓶颈,虽然近年来硬件设备性能大幅提升,但仍难以满足像深度学习模型这类计算密集型应用的需求,尤其针对特定领域应用,对设备的续航时间、响应速度、稳定性和可靠性均有较高的要求,这给算法设计造成了更大的困难.边缘计算的发展为解决此类问题提供了新的思路,相比于传统的云计算,边缘计算采用最近端服务策略,以最快速度响应任务请求从而满足任务的实时性需求.移动终端节点可以将计算任务全部或部分卸载到边缘云,以缓解本地计算压力, 实现功耗和处理速度的双重优化.这使得研究人员不但可以从算法设计角度,研究一种时效性强、准确度高、计算量小、能耗较低的目标追踪算法,还可以从应用部署的角度,通过研究计算任务分割、卸载、设备间通信以及数据融合等策略,进一步优化应用性能.

本文对基于计算机视觉的目标追踪算法的部署策略进行研究,其硬件环境主要为可穿戴设备或普通嵌入式设备,相比常规目标追踪算法的主要难点,应用设备还面临计算能力较弱、响应度和准确度要求较高的问题.本文通过研究基于边缘计算的任务分割和信息融合等策略,提升应用响应速度的同时,降低移动终端节点计算压力及能耗.

本文的主要贡献如下

1. 为嵌入式设备上深度学习模型的部署提供了新的思路. 针对移动嵌入式设备资源不足, 深度学习模型难以部署应用的问题, 引入边缘云对移动设备提供支撑, 从计算、通信、存储的角度对计算模型的部署策略进行优化, 采用终端节点与边缘云协同处理的方式进一步提升任务处理速度.

2. 针对特定应用场景, 为保证终端设备在较差网络环境下的独立作业能力, 提升系统整体的容错性和鲁棒性, 在移动终端节点部署轻量级目标追踪算法, 同时结合集成学习策略融合本地终端节点和边缘云端计算结果, 使移动设备具备网络环境自适应的能力.

3. 提出基于峰值置信度的目标追踪算法集成方式, 并通过响应图重建策略降低传输数据量.

本文组织结构如下:第 1 节对目标追踪和边缘计算的相关工作进行介绍;第 2 节详述所提出的目标追踪应用部署策略;第 3 节实验验证部署策略的有效性,并对实验结果进行分析;第 4 节对全文进行总结.

1 相关工作

1.1 目标追踪算法

目标追踪算法按照追踪目标和摄像机数量,可分为单目标单摄像机(STSC, Single Target Single Camera)、多目标单摄像机(MTSC, Multi Target Single Camera)、多目标多摄像机(MTMC, Multi Target Multi Camera),其中 MTMC 可以看成是 MTSC 与 ReID(Re-Identification)技术的结合.其主要流程一般为在第一帧给定追踪目标位置,通常为方形检测框,在后续数据帧中,算法对追踪目标进行跟随,同时给出算法计算得出的检测位置以及尺度.目前,单目标追踪算法大体可以分成两类,分别为基于相关滤波器的算法和基于深度学习的算法.

基于深度学习技术的单目标追踪算法,主要可以分为二类,一类将深度学习技术与相关滤波器相结合.如,Danelljan M 等人提出的 DeepSRDCF[1] 算法采用深度特征取代手工特征以提升模型性能. Ma C 等人提出的 HCFT[2]算法采用 VGG-Net[3]进行特征提取,集成从不同特征图谱所学习到的滤波器作为最终模型,以利用多层特征,取得了更优的追踪性能.另一类则完全采用深度学习相关技术进行追踪,如 Tao R 等人[4]采用孪生神经网络(Siamese network)比较追踪目标与候选目标之间的相似性,进而确定目标位置.

基于孪生神经网络的目标追踪算法是近来最主要的研究分支,主要原因在于此类方法兼顾了处理速度和算法性能.Bertinetto L 等人提出 SiamFC[5]算法,该方法创新性的将追踪问题视为相似学习问题,避免了深度学习模型的在线更新,在速度和性能两方面均取得了非常不错的效果.SiamRPN[6],通过引入物体检测领域的区域建议网络(RPN, Region Proposal Network)进一步提升性能.针对将 SiamFC 和 SiamRPN 的主干网络替换为更深层结构时,算法并未取得更好的性能的问题,Li B 和 Zhang Z 等人从不同的角度探究其原因,分别提出了 SiamRPN++[7]和 SiamDW[8]. Wang Q 等人在原有网络基础上添加 Mask 分支提出 SiamMask[9],完成目标追踪任务的同时实现物体分割,在一般场景下效果显著,但对于遮挡等问题鲁棒性较差.针对不同算法的优化,一方面,可以从算法模型的角度进行改进,如改进模型架构设计,或采用集成学习方法,结合不同种算法优点,构建性能更加优良的算法模型.另一方面,可以从应用部署的角度,通过研究计算任务分割、卸载、设备间通信以及数据融合等策略,进一步优化应用性能.本文以 STSC 算法为主要研究对象,从应用部署优化的角度,对目标追踪算法在移动设备上的部署策略进行研究,部署策略也可进一步扩展至其他类型算法.

1.2 边缘计算技术

新型网络任务和场景,如自动驾驶,安防监控等,对于网络延时和可靠性安全性等方面的高要求使得传统网络架构难以应对,边缘计算技术应运而生.边缘计算主要包含虚拟化、云计算和软件定义网络等关键技术. Shi W 等人[10]将“边缘”定义为数据源与云数据中心之间的任何计算资源和网络资源.对于其优势和必要性,Hu W 等人[11]通过实验进行了相关验证,并通过实验证明移动设备盲目卸载计算任务到云可能导致更低的性能和更高的能耗.

目前,边缘计算已在诸多领域中得以应用.Garg S 等人[12]将边缘服务器作为中间接口,辅助车辆与云端数据中心间通信,减少了终端节点访问时间和网络拥塞.Sheng Z 等人[13]将无线声音传感器网络与边缘计算相结合,使得成本和能耗更低.Lai C F 等人[14]利用边缘云实现并行计算以提高监管系统对工业设备的识别效率.Muhammad G 等人[15]将边缘计算应用于智慧医疗框架.相比传统网络架构,边缘计算采用降低服务器和用户间距离的方式减少了网络响应时间,同时降低了数据传输功耗和网络堵塞时间[16].然而,此类应用框架仅利用了边缘云的优势,并未针对应用模型进行更近一步的拆分.与此不同,本文针对应用模型本身的计算任务进行更细致的划分,从模型部署的角度进行更进一步的优化.

边缘云架构一直是该领域的主要研究方向.Tong L 等人[17]将边缘云更加细化,将边缘云层设计为一种树状的层次结构.允许不同服务器层对峰值负载进行聚合,使得云资源利用率更高. Yao C 等人[18]从不同设备以及接入点的角度提出多层边缘计算框架 EdgeFlow,通过权衡设备的资源占用情况以及通信状态,将不同任务以最佳方式分配给每一层. Chia-Wei T 等人[19]提出一种基于网关的边缘计算服务模型,可实现资源按需分配.相比以上几种边缘云架构,在特定应用场景下的平台架构设计中,将计算能力较强的移动终端节点上移边缘云层是更为合理可靠的.例如,野外环境下,设备工作环境可能较差,网络延迟可能使得终端节点设备与边缘云无法畅通连接,为此,将比穿戴式设备计算能力更强的节点设备作为其他设备的边缘云节点,移动终端节点可自由选择任务卸载位置,从而降低网络环境对计算任务的影响.

针对边缘云架构中的任务卸载、资源分配及数据传输等问题,不同学者对此进行了研究探索.Sun Y 等人[20]提出一种自适应任务卸载算法,相比基于置信上限的学习算法,平均延迟降低了 30%.针对如何满足计算任务延迟条件且保证系统成本最小化的问题,Zhang Y 等人[21]提出一种两阶段任务调度成本优化算法,在满足所有任务延迟的同时,使系统成本最小化.针对边缘服务器的过载问题,Fan Q 等人[22]设计了一种基于应用感知的工作负载分配方案,通过为用户不同类型的请求分配合适的计算资源以最大限度地减少响应时间.Du W 等人[23]将边缘计算的资源分配问题定义为一个随机优化问题进行求解,取得了较好的效果,然而该算法并不考虑资源共享的公平性问题.Peiyuan G 等人[24]采用博弈论的观点对边缘计算环境进行了分析,证明其中存在纳什均衡.Lin B 等人[25]针对边缘数据中心间的数据传输问题,提出了一种自适应离散粒子群优化算法(GA-DPSO)以缩短数据传输时间.Ren Y 等人[26]为更好的适应边缘计算应用,开发了一种基于微内核的操作系统 EdgeOS.然而,目前针对模型计算任务划分的研究相对较少,尝试针对基于深度神经网络的算法模型进行更细致的计算任务划分,

结合边缘计算,解决深度学习技术在嵌入式领域应用性较差的问题.

1.3 目标追踪系统应用场景分析

追踪算法处理流程可分为多个阶段,且算法具有基本相同的处理步骤,这为边缘计算的应用提供了可行性,通过追踪任务进行分割,采取不同阶段不同节点计算的策略, 针对不同处理阶段,可以将相同计算任务进行特殊优化,如采用 FPGA 等设备进行预处理阶段中图像剪裁、图像增强、傅里叶变换等操作,将深度特征提取等运算卸载至边缘云.以减小本地计算压力及功耗.

基于边缘计算的单目标追踪系统在一定程度上与分布式机器学习系统类似,后者通常包含数据和模型划分、单机优化、通信以及模型和数据聚合等部分.将训练样本通过随机采样或置乱切分等方式进行划分,并对模型进行横向、纵向或随机划分后,分配至不同工作节点,采用单机优化策略进行子模型训练,期间通过通信策略进行信息同步,最后采用模型和数据聚合算法对训练好的模型进行集成聚合.与此类似,基于边缘计算的目标追踪系统主要包含任务分割、目标追踪、通信以及信息融合等部分.其中,终端节点和边缘云服务器分别部署目标追踪算法,通过任务分割策略决策计算任务处理节点,通信方案决策信息收发内容及通信步调等,计算任务卸载至云端运算后,通过信息融合策略对终端节点和边缘云两侧计算结果进行融合.

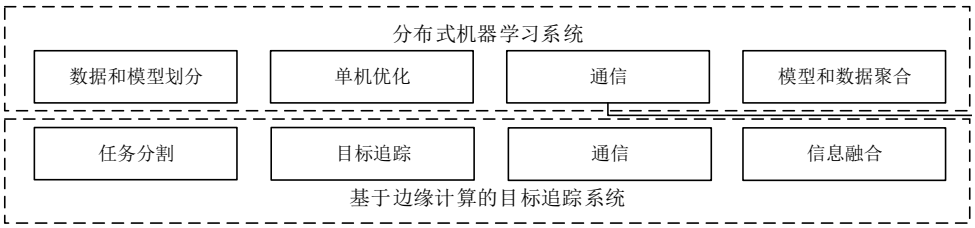


Fig.1 Comparison of distributed machine learning system and target tracking system

图 1 分布式机器学习系统与目标追踪系统对比示意图

应用场景:本文所研究的部署策略主要针对工作于野外环境下的穿戴式设备或救援机器人等类似移动设备,如救援机器人等.野外环境下,特定用户人群,如搜救或抓捕人员等,通过配备穿戴式设备辅助追踪特定人员或目标.小组成员间通过信息传递,可进行多摄像机单目标连续追踪.通过保存追踪目标在此过程中的运动轨迹及外观信息可进一步对目标进行行为分析或融合其他信息进行更高层信息提取.

2 目标追踪应用部署策略研究

2.1 边缘云网络架构

边缘云网络架构主要面向野外环境下穿戴式设备或救援机器人等类似移动设备.利用边缘云解决终端节点对应用程序计算压力及功耗等方面的限制.网络架构总体分为终端节点、边缘云和云端数据中心三层.终端节点层主要为接入边缘云的移动设备,每个节点部署应用包含目标追踪、计算任务分割、信息融合和运动检测等多个部分.边缘云层主要为边缘云资源部署层,包含边缘计算服务器和文件服务器等硬件资源设备,部署完成终端节点计算任务所需的必要程序,主要包含边缘云目标追踪、信息融合、信息管理和通信等部分.云端数据中心为与移动终端节点相距较远的数据中心,该层仅与边缘云进行通信.相比 1.2 节中所分析的三种层次边缘云架构,如将云端资源下移边缘端或将边缘云层继续拆分,在网络架构设计中,将计算能力较强的移动终端节点上移边缘云层.野外环境下,设备工作环境可能较差,网络延迟可能使得终端节点设备与边缘云无法畅通连接,为此,将比穿戴式设备计算能力更强的节点设备,如笔记本电脑、台式机或小组内其他计算能力较强的设备等,作为其他穿戴式设备的边缘云节点.穿戴式设备或其他移动终端节点可自由选择任务卸载位置,从而降低网络环境对计算任务的影响.

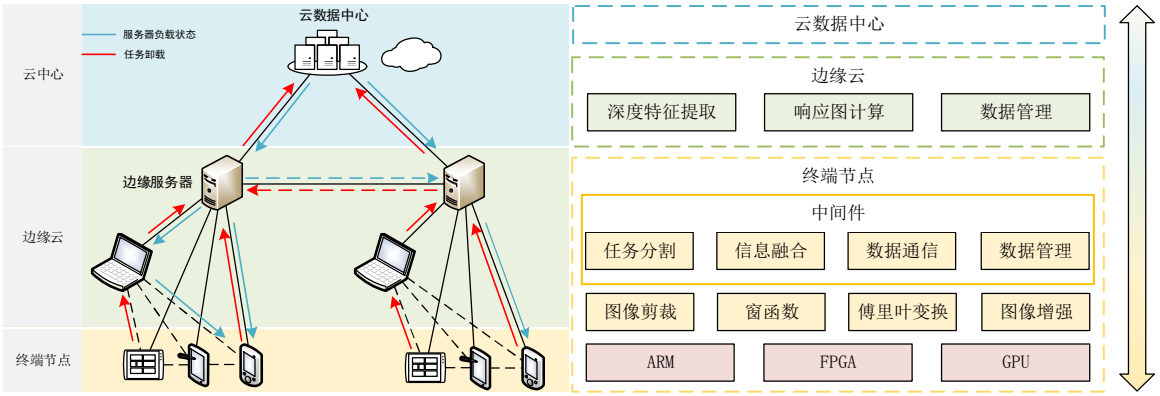


Fig.2 Edge cloud network architecture and software architecture

图 2 边缘云网络架构及软件架构

为详细说明部署策略,搭建目标追踪系统,如图 3 所示,主体为基于计算机视觉的目标追踪算法,输入数据为视频流数据,具体为视频帧图片.任务分割策略根据本地计算负载状态、如设备温度、CPU 利用率、内存占用率以及能源状态等,判定是否进行本地计算,与此同时,根据网络状态和边缘服务器资源占用情况,判断是否将发送边缘云进行计算,同时决定计算任务分割点,用以最终决策终端节点和边缘云的任务计算量. 由于在野外环境下,网络情况难以长期保证,且设备具体应用环境较为复杂,如搜救机器人时常工作于废墟或矿道等恶劣环境下,通信情况难以保证,且此时任务失败将造成严重后果.因此,当移动终端节点与边缘云难以保持畅通连接时,终端节点应部署目标追踪算法使其具备独立作业能力,否则将直接导致追踪任务失败,造成严重后果.终端节点可采用基于相关滤波器的目标追踪算法进行部署,以满足移动节点计算能力较弱且能耗要求较高的特点.

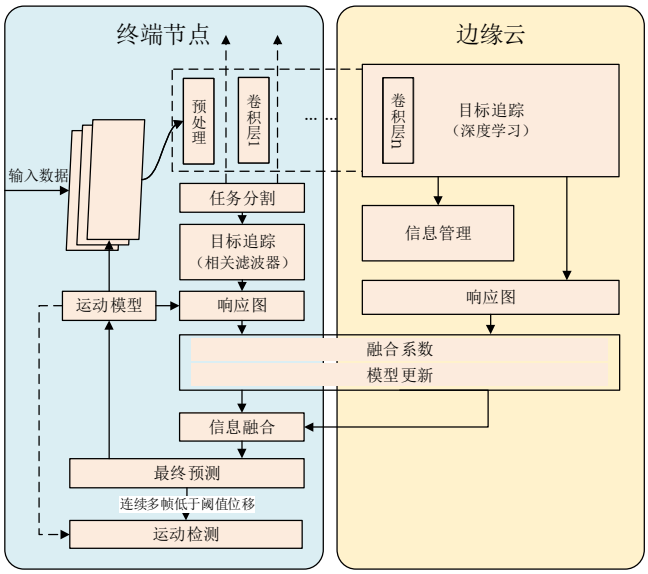


Fig.3 Schematic diagram of target tracking system software framework

图 3 目标追踪系统软件框架示意图

边缘云采用基于深度学习模型的追踪算法以提升整体追踪性能.根据算法计算得出的响应图(Response Map)确定算法融合系数,并确定此时终端算法模型是否进行更新.信息融合算法通过融合终端节点和边缘云返回的计算结果对追踪目标的具体位置进行最终预测.此外,终端节点采用卡尔曼滤波器对追踪目标的运动情况进行建模,另一方面,采用运动模型对追踪目标的位移阈值进行动态设定,当连续多帧目标位移均超出该阈值范

围时,启动运动检测以确定是否存在运动物体,以此进一步降低终端计算压力,避免不必要的计算.后文对各部分算法和具体策略进行详细介绍.

任务分割策略中,后续实验采用两种分割策略,即神经网络模型完整卸载和部分卸载.

(1) 模型完整卸载策略:主要将算法的特征提取部分卸载至边缘云进行计算,本地终端节点负责数据的预处理和后处理.其中数据预处理部分的计算任务,主要包括部分与输入图像数据相关的超参数的计算;以前一帧追踪目标的位置为中心,剪裁不同尺寸的图像以供后续尺度估计计算等.后处理部分主要包括,根据边缘云返回的响应图,对追踪目标的尺度及具体估测位置进行计算等;

(2) 模型部分卸载策略:相比将特征提取任务完全卸载至边缘云计算,将卷积神经网络模型中第一部分,即第一个卷积层、批标准化层(BN, Batch Normalization)及池化层分配至终端节点计算.

2.2 目标追踪算法

由于移动设备工作环境多样化,难以保证网络状态,因此当设备无法与边缘云畅通连接时,应保证终端节点仍具备独立作业能力,避免任务失败造成严重后果.基于以上原因,在终端节点部署计算压力较小的目标追踪算法.另一方面,由于算法通常专注于解决具体任务一个方面的问题,难以对任务的全部问题进行优化,因此,通常采用集成多个算法的方式以结合不同算法的优点,进而构造一个性能更加优良的算法模型以满足实际需求.例如基于计算机视觉的目标追踪算法,其主要难点包含光照、变形、尺度变化、背景杂乱等多个问题,不同算法通常针对某一问题进行设计优化,因此,实际应用时,通常采用集成的方式以获得更好的追踪性能.集成学习(Ensemble Learning)通过结合一系列算法模型以获得一个性能更好的算法模型,该策略在工程部署中得到广泛应用.在目标追踪过程中,可将移动终端节点的相关滤波器算法与边缘云的深度学习算法进行集成,以获得更加稳定且性能更加优良的算法模型.研究过程中,选用以下两种算法进行部署研究,具体模型可更换.

(1) 终端节点目标追踪算法.

终端节点采用基于相关滤波器(CF, Correlation Filter)的目标追踪算法进行部署以满足移动终端节点计算能力较弱且对续航能力要求较高的特点,算法模型采用 DCF[27].

DCF(Dual Correlation Filter)在 CSK[28]基础上进行改进,利用循环矩阵进行循环采样以解决稀疏采样问题,并引入多通道特征,使得性能相比以往目标追踪算法大幅提高.DCF 采用岭回归建模,通过最小化采样数据的计算标签与目标真实位置之间的距离求解函数,并引入循环矩阵进行循环采样.DCF 采用线性核(Linear Kernel),根据当前图片数据样本 x 计算核函数 $k(x,x)$和参数 α 值,当下一张图片数据 z 传入时,计算 $k(z,x)$和 $f(z)$,取 $f(z)$实部作为响应图,其中最大值位置为算法预测目标位置.

DCF 算法的主要优点在于其处理速度,适用于计算能力较弱的移动设备.在终端节点部署该类算法,用以保证设备无法连接边缘云时仍具备独立作业能力,避免任务失败.

(2) 边缘云目标追踪算法

基于边缘云计算资源丰富,处理速度快等特点,算法部署无需过多考虑硬件资源开销,可采用基于深度学习模型的目标追踪算法进行部署以满足追踪任务的性能需求,算法模型采用 SiamFC[5].

全卷积孪生神经网络(SiamFC, Fully-Convolutional Siamese Networks)创新性的将目标追踪作为一种相似性学习问题(Similarity Learning),模型采用大量数据进行离线训练后,实际部署进行追踪时,无需对模型进行更新,从而避免了追踪过程中的复杂计算,使得算法性能满足实时追踪任务要求.通过相似性度量函数对样本图片和候选图片之间的相似度进行计算可以对两者间的相似程度进行衡量,并返回对相似程度进行评分,相似度越高则分数越高.算法采用全卷积孪生神经网络作为相似性度量函数.该算法在处理速度和追踪性能上均有取得了良好表现,也由此引起众多学者基于全卷积孪生神经网络的进一步研究改进.由于算法采用相似性学习的角度解决追踪问题,算法可采用大量视频序列对模型进行离线训练,因此针对特定场景下的目标追踪任务可以采取预先收集相似场景数据以更好的训练模型,从而取得更好的性能.

2.3 任务分割策略

基于边缘计算的追踪系统,常用任务卸载策略为将全部计算任务卸载至边缘云.另一种策略,以算法HCFT[3]为例,由于根据多个卷积层输出的特征图学习多个不同的相关滤波器,因此可根据设备负载状态及网络环境,动态判定所学习的滤波器个数.与此不同,本文根据算法的处理阶段对计算任务进行划分,如预处理及特征提取计算量较小,而此时网络延迟较大,则可将该部分计算任务本地运算,且由于搜索区域相比整张图片区域更小,因此处理后信息传输量更小,降低了网络传输延迟.此外,由于在终端节点和边缘云服务器分别部署不同的目标追踪算法,通过任务分割策略决策计算任务本地计算、卸载至边缘云计算或同时计算后融合,最后通过信息融合策略,对不同种算法计算结果进行集成.

任务分割策略主要根据本地计算节点负载情况、硬件设备利用率、网络延迟等状态对计算任务进行划分,将计算任务部分卸载或全部卸载至边缘云,以达到能耗、计算速度和准确度的组合优化目的.计算任务拆分时,一方面考虑本地处理后,数据量更小,从而降低网络传输延迟;另一方面,在网络处于高延迟状态时,通过本地继续处理后续计算任务,从而避开当前高延迟时段,减少不必要的等待延迟.

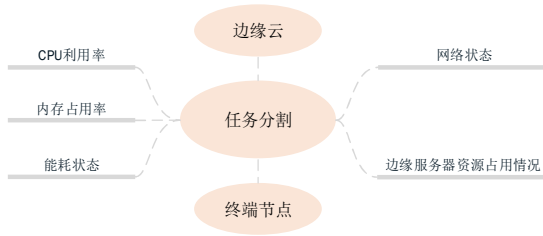


Fig.4 Schematic diagram of computing task segmentation strategy

图 4 计算任务分割策略示意图

本文研究内容在一定程度上可视为多约束条件下的资源分配问题,设备在当前工作环境下,除了以上约束, ,还受到计算任务本身的约束,在目标追踪应用中,为保证追踪的实时性,每帧图像数据的处理时间一般应保持在50ms以下,即每秒处理数据应高于20帧.此外,资源优化分配时,还应考虑任务性能的约束,即目标追踪整体模型性能应保持在一定范围内.

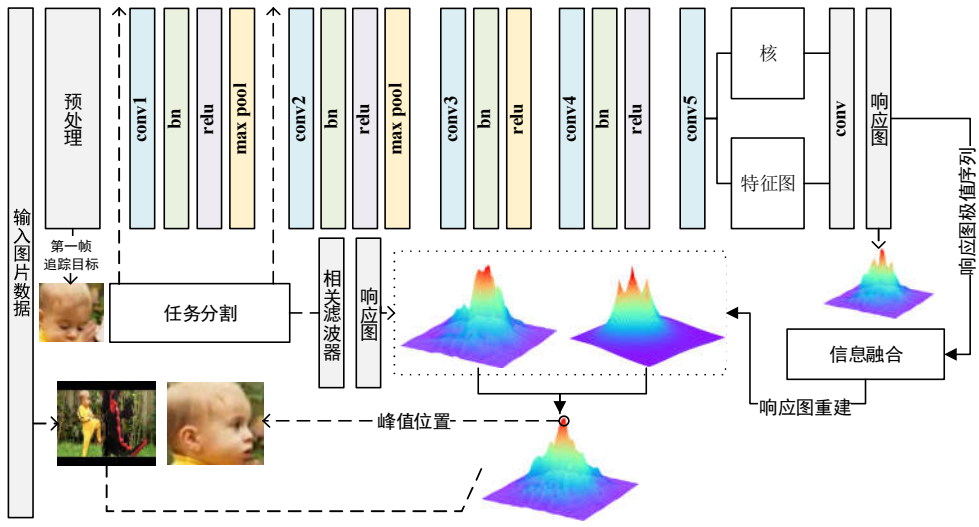


Fig.5 Schematic diagram of the calculation process of the task (picture is OTB100[29] DragonBaby)

图 5 计算任务处理流程示意图(图片为 OTB100[29] DragonBaby)

经过环境信息探查后,对所得信息进行融合后判定计算任务分割点,决策本地终端节点和边缘云计算任务量. 计算任务主要处理流程如图 5 所示. 计算任务可从数据预处理阶段和特征提取之间进行划分,即将神经网络完整卸载至边缘云;或从神经网络模型不同层之间进行划分,即将神经网络模型分割卸载至边缘云.通过任务分割策略判定本地终端节点进行预处理及裁剪运算,或继续进行第一层或几层特征提取,处理后数据上传边缘云进行下一步计算.边缘云计算获得响应图后,返回响应图中极值序列,数据交由本地,通过信息融合策略进行响应图重建.融合终端节点目标追踪算法处理结果后,最终判定目标位置.边缘云仅负责特征提取及响应图相关计算任务,本地终端节点负责数据预处理、浅层特征提取以及获得极值序列后进行响应图后重建、尺度判定、参数计算更新等计算任务.

2.4 信息融合策略

集成学习(Ensemble Learning)目前已成为工程应用中经常采用的策略,用以聚合不同算法的优势.本文通过评测响应图波动程度来评价模型对当前预测的目标位置的置信程度,根据峰值置信度指标对不同算法计算所得的响应图谱进行度量,由此确定具体融合系数.

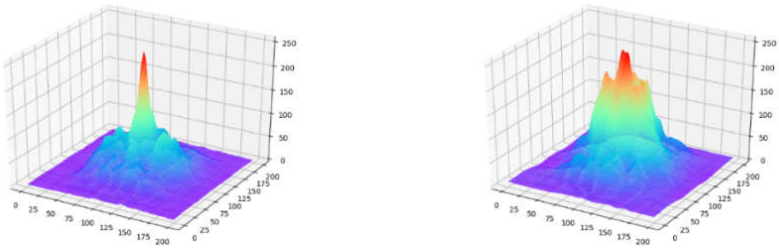


Fig.6 Filter response map (left) higher confidence (right) lower confidence

图 6 滤波器响应图 (左)置信度较高 (右)置信度较低

响应图预处理时,归一化其取值范围到[*0*, *M*].

$$peak\_confidence := \frac{Var(D_\theta) - Var(D_{\theta'}) + \delta}{Var(D_\theta) + \delta} \quad s.t.\ peak\_confidence > \theta_{pc} \tag{1}$$

其中 *Var()*表示方差,D_θ 为响应图中极值数据点集合,$D_{\theta'}$ 表示去除响应图中最大值后极值点集合,且集合 D_θ 中滤除小于阈值 θ_{pc} 的极值,以增强鲁棒性,超参数 θ_{pc} 预先设置. ε 为超参数,用于避免运算中数值溢出而预先定义的极小值.滤波器仅在峰值置信度高于阈值时进行更新.

此外,由于响应图传输时间对算法总体处理时间影响较大,算法仅传输响应图中高于一定阈值的极值集合以降低信息传输量,终端节点通过回传极值信息,对响应图进行重建.

$$G(x,y) = \sum_{i=1}^{l} V_i g_i(x,y) \quad s.t.\ g_i(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{\left((x-x_i)^2+(y-y_i)^2\right)}{2\sigma^2}} \tag{2}$$

其中 *G(x, y)*为重建后曲面,$g_i(x, y)$为以第 *i* 个极值点坐标为中心的二维高斯曲面,V_i 为对应极值.

由于高斯函数的指数运算使得计算压力较大,使得上述策略无法满足实时性要求,因此进一步优化,采用平移策略代替创建二维高斯函数的相关运算.

$$G(x,y) = \sum_{i=1}^{l} V_i P_g(x_i,y_i) \quad s.t.\ g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{\left(x^2+y^2\right)}{2\sigma^2}} \tag{3}$$

其中 *G(x, y)*为重建后曲面,*g(x, y)*为预先创建的以(*0*, *0*)坐标为中心的二维高斯曲面,V_i 为对应极值,$P_g(x_i, y_i)$为将

曲面 $g(x, y)$平移(x_i, y_i)后的高斯曲面.

2.5 运动检测方案

由于终端设备对应用程序的计算压力及功耗要求较高,且当追踪算法丢失目标时,继续计算将造成严重的资源浪费,然而追踪算法正确跟踪目标或已丢失目标通常是难以判定的,因此采用运动模型对追踪目标的运动状态进行建模,通过运动模型动态设定位移阈值范围,当追踪算法预测位置连续多帧超出此阈值范围时,初步判定丢失目标,启动运动检测算法检测是否存在运动物体以确定是否已丢失目标.后续可辅助算法在丢失目标时进行重检测.运动检测算法采用帧差法进行部署以满足计算压力小、运算速度快以及低能耗的要求.此外,建立运动模型以针对不同追踪目标和该目标在追踪过程的不同阶段自适应设定位移阈值.

(1)运动模型

对于目标追踪算法,物体的运动状态对于模型追踪具有积极影响,尤其在追踪过程中存在遮挡和运动模糊等情况时.因此,对追踪目标运动状态建模以进一步提升性能.此外,由于目标在追踪过程中的动态变化,使得对目标位移设置固定阈值以判断滤波器预测结果是否处于正常范围内并不合适.基于以上原因,采用运动模型对目标位移阈值进行动态设定以适应目标在追踪过程中的动态变化.

运动模型采用卡尔曼滤波器(Kalman Filter)进行建模,阈值设定如下所示,

$$\theta_x = \rho_\theta x^h \quad s.t.\, x^h = \rho_x x^{h-1} + (1-\rho_x)(\hat{x}_k - \hat{x}_k^-) \tag{4}$$

其中,θ_x 为为阈值位移,x^h 为运动模型估测位移的滑动平均值,ρ_θ 为系数. $\hat{x}_k$和$\hat{x}_k^-$分别为追踪算法和卡尔曼滤波器所估测的目标位移.

(2)运动检测算法

当追踪算法连续多帧预测目标位移超出阈值范围时,采用运动检测算法对输入数据进行运动检测以进一步降低模型能耗,提升算法处理速度,减少不必要的计算.常用运动检测算法有帧差法、光流法、背景减除法、ViBe 算法等.其中帧差法具有简单快速,对于光照环境不敏感以及对于动态环境适应性较强等特点,适用于终端节点硬件环境以及应用程序运行环境.

帧差法通过对相邻两帧图像做差分运算以检测运动物体,其主要理论依据在于,相邻两帧或三帧进行灰度值差值运算后,运动物体由于灰度值变化将会产生灰度残留,而静止物体则由于灰度不变被差值运算去除.算法主要步骤:连续输入三帧图像数据,前两帧和后两帧分别计算灰度差值后,结果进行按位与运算得出结果.运动检测时,获得三帧差法结果后,在目标追踪算法预测位置,取目标大小区域,通过对该区域内数值进行求和并与阈值比较以确定该处是否存在物体运动.

Fig.7 Schematic diagram of motion detection results (OTB100[29] Bird1)

图 7 运动检测结果示意图(OTB100[29] Bird1)

2.6 信息管理方案

信息管理主要用于存储、分发以及更高层信息提取等任务.边缘云对目标在追踪过程中的状态进行保存,记录物体运动路径以及目标活动,历史记录信息主要用于特定场景下决策判断或后续进一步的信息提取.例如,对于搜救机器人,通过分析运动记录,可以确定已搜索区域,从而辅助判断下一步搜救区域；对于其他追踪目标,可根据记录信息对目标的行为习惯或行动意图进行判断.此外,记录信息可结合其他应用信息进行更高层信息融合或发送其他终端节点以供其他后续操作.

3 实验

3.1 实验环境

边缘服务器采用 12 核 Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz,GPU 采用 GeForce RTX 2080,显存 12G. 终端节点采用平板电脑,Intel(R) Core(TM) i5-8250U CPU @ 1.6GHz,8G RAM,CPU 处理.软件环境,服务器系统 Ubuntu,终端系统 Windows7,编程语言 Python3,CUDA10.0,PyTorch1.1.0.测试工具 GOT-10k[30],测试数据集为 OTB100[31].

3.2 目标追踪系统实验

为验证任务分割策略的有效性,首先测试计算任务完全由终端节点计算和完全卸载至边缘云计算时,完成相同计算任务所需时间,以供对比分析.

(1)计算任务完全由终端节点处理

计算任务完全由本地终端节点计算,处理时间如图所示.计算任务主要包括数据预处理,深度特征提取,响应图计算以及后处理等阶段.

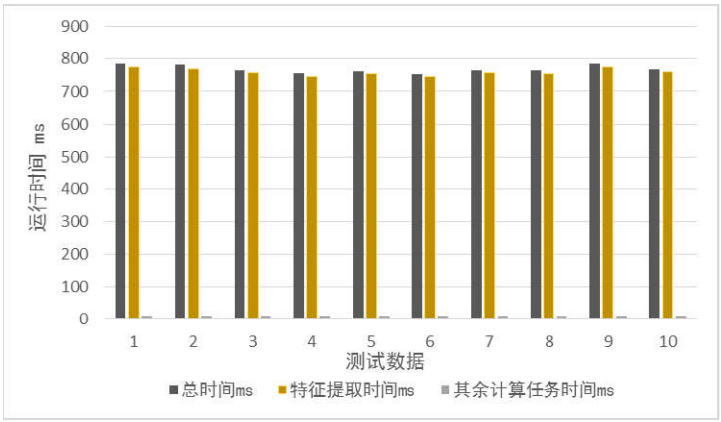


Fig.8 Terminal node calculation task processing time (calculation task completely local calculation)

图 8 终端节点计算任务处理时间(计算任务完全本地计算)

计算任务完全由本地终端节点运行,从测试结果中可以看出,采用深度学习网络模型进行特征提取成为主要性能瓶颈.其余计算任务处理时间为 10-15ms 左右.

(2)计算任务完全由边缘云处理

计算任务完全卸载至边缘云处理时,终端节点仅将本地获取到的图片数据上传边缘云,所有计算任务均由边缘云完成. 嵌入式终端计算负载仅为摄像头图像获取、数据发送/接收,目标区域绘制及显示.

实验中,边缘云模型架构采用百度边缘计算平台 OpenEdge .该平台可将云计算能力拓展至用户现场,提供临时离线、低延时的计算服务,包括设备接入、消息路由、消息远程同步、函数计算、设备信息上报、配置下发等功能. 通过 OpenEdge 和 智能边缘 BIE（Baidu-IntelliEdge）协同部署,可在云端进行智能边缘核心设备的建立、服务创建、函数编写,然后生成配置文件下发至本地运行,达到云端管理和应用下发,边缘设备上运行

的效果,满足边缘计算应用场景.

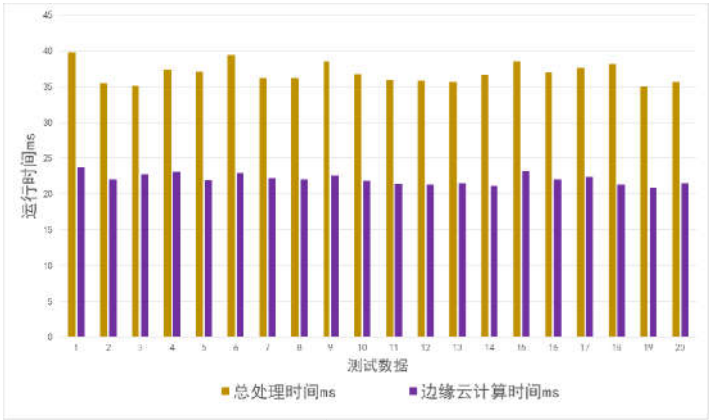


Fig.9 Edge cloud computing task processing time (calculation task completely unloads edge cloud computing)

图 9 边缘云计算任务处理时间(计算任务完全卸载边缘云计算)

计算任务完全卸载至边缘云计算,其总计算时间为 40ms,边缘服务器用于计算任务处理时间约为 23ms,其余时间主要为信息传输时间.实验中采用的数据为 OTB100[29]数据集中 DragonBaby 图片序列,图片分辨率为 640*360, 从实验结果可以看出,信息传输时间对于总处理时间仍存在较大影响,当终端节点获取到图片数据较大,分辨率更高时,将产生更高的传输延迟,严重影响追踪性能.

(3)终端节点与边缘云协同处理

计算任务完全本地计算时,嵌入式终端计算负载为算法全部计算任务.计算任务完全卸载至边缘云计算时,嵌入式终端计算负载仅为摄像头图像获取、数据发送/接收,目标区域绘制及显示.

协同处理时,计算任务部分卸载至边缘云端.本地嵌入式终端负责图像数据的获取、预处理以及神经网络模型的前几层计算.本地计算任务量介于完全本地计算和完全卸载云端计算之间.通过将部分计算任务下移本地,使嵌入式终端和边缘云协同处理,以达到降低任务整体响应时间的目的.

1) 计算任务划分后算法性能测试

为验证计算任务分割策略对于追踪算法性能的影响,首先对任务分割后算法进行性能测试,测试实验中,采用模型部分卸载策略,测试结果如图 10 所示.实验结果表明,计算任务分割后,算法整体性能并未受到影响,其性能与未分割的原始算法相同,即计算任务划分对算法性能无影响,仅将相同计算任务划分至不同设备进行处理.

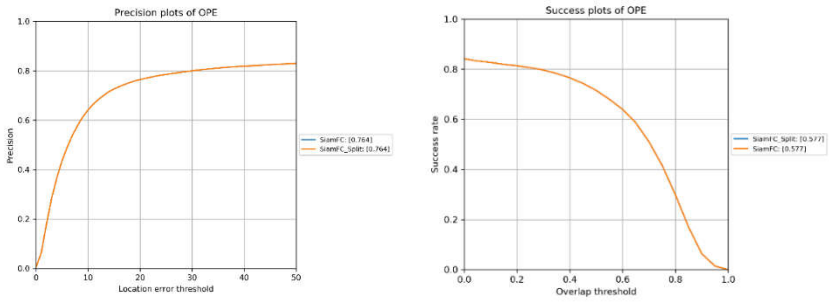


Fig.10 SiamFC and SiamFC_Split algorithm performance test chart (Model partial offload)

图 10 SiamFC 和 SiamFC_Split 算法性能测试图(模型部分卸载)

2) 神经网络模型完整卸载策略

该策略计算任务划分中,主要将算法的特征提取部分卸载至边缘云进行计算,本地终端节点负责数据的预处理和后处理.其中数据预处理部分的计算任务,主要包括部分与输入图像数据相关的超参数的计算；以前一

帧追踪目标的位置为中心,剪裁不同尺寸的图像以供后续尺度估计计算等.后处理部分主要包括,根据边缘云返回的响应图,对追踪目标的尺度及具体估测位置进行计算等.

由于目标追踪任务对实时性要求较高,且追踪目标的运动和变化通常具有平滑特性,至少为局部平滑,即通常情况下,相邻两帧中,物体位置相近且外观相似,因此,通常基于上一帧位置截取 1.5~2 倍大小目标区域进行检测,从而避免对整张图像进行运算,提升了模型的计算速度.此外,目标追踪算法处理追踪目标尺度变化的一个主要方法,是以上一帧中目标所在位置为中心,在当前帧中提取多个不同尺度的图片区域进行检测匹配,选择响应程度最高的尺度作为追踪目标.因此,可以通过调节预处理操作中截取区域的大小和数量,以降低信息传输量,从而降低信息传输时间.

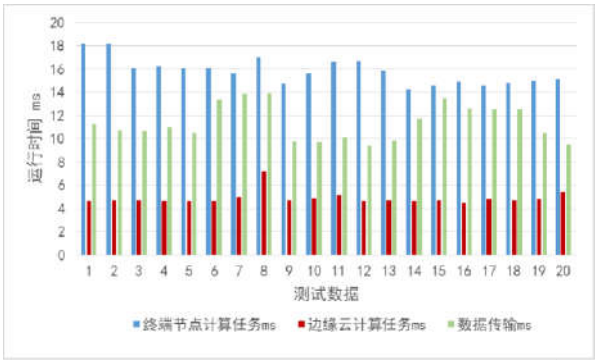


Fig.11 Computing task processing time in complete offloading strategy of neural network model

图 11 神经网络模型完整卸载策略中计算任务处理时间

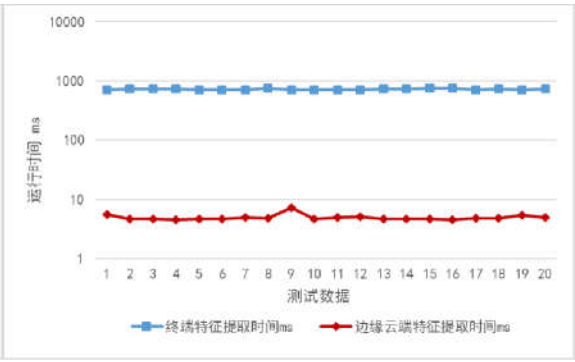


Fig.12 Feature extraction time comparison chart

图 12 特征提取时间对比图

相同计算任务进行划分后,终端节点与边缘云计算任务处理时间及特征提取时间如上图所示,从图中可看出,利用边缘云处理计算压力较大的特征提取任务,可以有效降低计算任务处理时间.

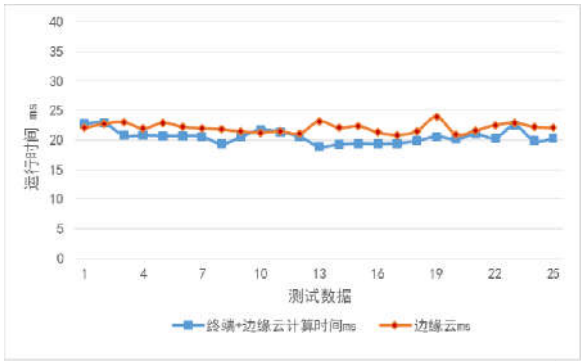


Fig.13 Same calculation task total processing time

图 13 相同计算任务总处理时间

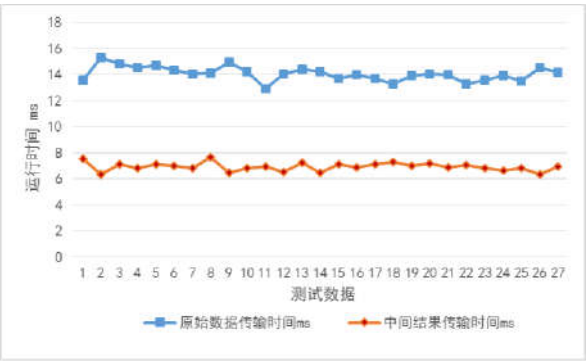
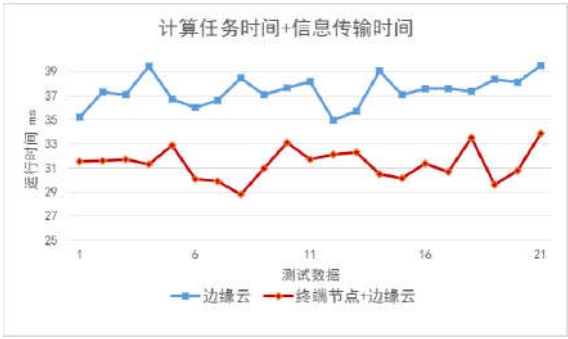


Fig.14 Information transmission time

图 14 信息传输时间

上图 13 为相同计算任务的总处理时间,图 14 为数据传输时间.实验结果表明,将部分计算压力较小的计算任务本地运行,与所有任务完全卸载至边缘云计算,总处理时间基本相同.此外,由于原始数据经预处理后,中间结果相比原始数据量小,因此传输时间更低.

相同计算任务处理及信息传输总时间如图 15 所示,任务分割后,终端节点与边缘云协同处理时间约 31ms,相比完全卸载至边缘云处理(约 38ms),可以更快的完成相同计算任务,速度提升超过 15%. 追踪任务总体计算时间测试结果如图 16 所示,可以看出,任务分割后总体处理时间明显低于完全由终端节点处理所用时间.



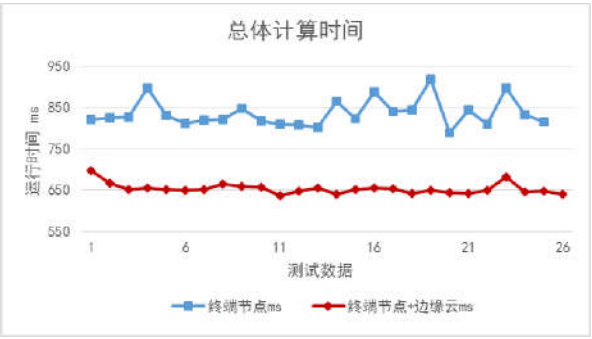


Fig.15 Calculation task and total information transmission time Fig.16 Track task overall calculation time

图 15 计算任务与信息传输总时间 图 16 追踪任务总体计算时间

对任务处理时间进行分析,发现主要瓶颈在于数据转换时间.该问题由于实现方式导致.数据转换主要将深度学习框架下的数据类型向基础数据类型转换以适配应用框架的信息传输要求,时间延迟主要包括数据类型转化、信息打包及解析时间.后续将针对该问题进行改进优化.

3) 神经网络模型部分卸载策略

相比将特征提取任务完全卸载至边缘云计算,模型部分卸载策略中,将卷积神经网络模型中第一部分,即第一个卷积层、批标准化层(BN, Batch Normalization)及池化层分配至终端节点计算.计算结果如图 17 所示.

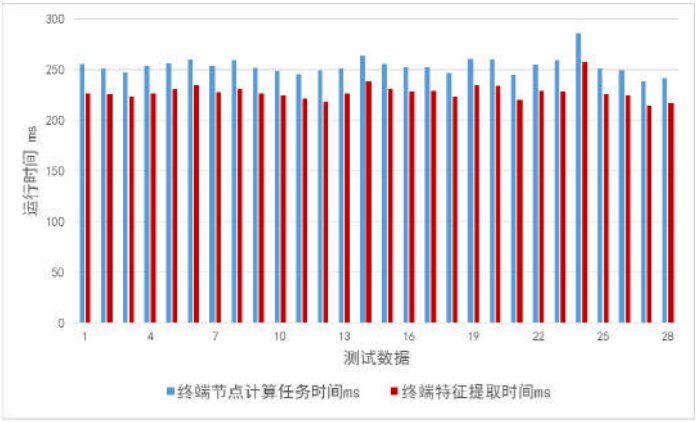


Fig.17 Neural network model partial offloading strategy terminal node computes task processing time

图 17 神经网络模型部分卸载策略终端节点计算任务处理时间

从图中可以看出,对于不搭载 GPU 的终端节点设备,即使处理单层神经网络模型卷积运算,仍存在较大计算压力.然而对于目前多数已搭载轻量级 GPU 或 NPU 的移动设备或可穿戴设备,将神经网络模型部分计算任务分配至本地 GPU 计算,以避开网络高延迟时段,仍是值得考虑的部署策略.

(4)运动检测及响应图重建策略测试

为验证运动检测和响应图重建策略对于整体处理时间的影响,对两种策略进行实验测试.

实验测试结果如图 18 所示,从运动测试结果可以看出,运动检测时间较短,仅需 2ms 左右.响应图重建测试中,对应 30 个极值点,采用指数运算进行响应图重建需要耗时 679ms,采用平移重建仅需 2.3ms,速度提升 300 倍.为提升算法鲁棒性,同时降低信息传输量以保证数据传输低延迟,对响应图中的极值点进行阈值过滤,仅保留高于一定阈值的极值点进行传输,因此极值点数量一般不会超过 30,从实验结果可以看出,重建策略对整体性能几乎无影响.

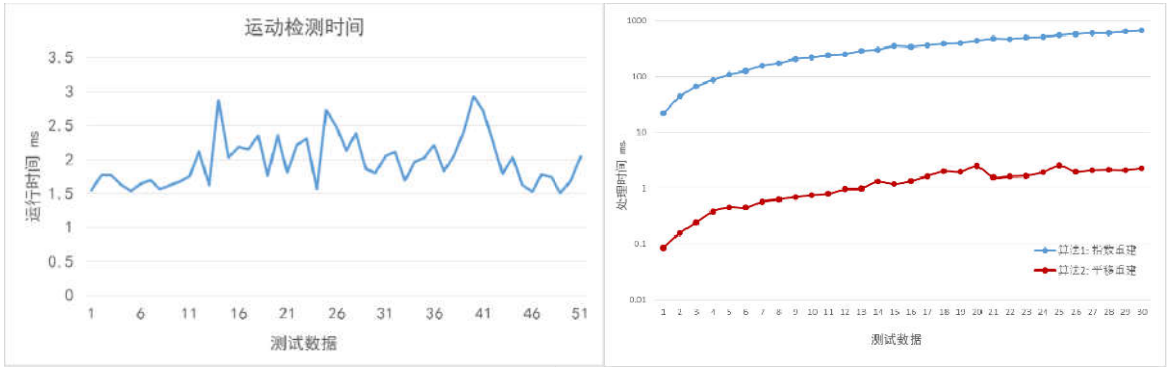


Fig.18 Motion detection time(left) Response graph reconstruction time (right), algorithm 1 (blue) indicates reconstruction by exponential operation, and algorithm 2 (red) indicates translation reconstruction

图 18 运动检测时间(左) 响应图重建时间(右),算法 1(蓝)表示通过指数运算重建,算法 2(红)表示平移重建

(5)终端节点计算资源占用率测试

嵌入式端的资源消耗与任务分割策略密切相关.实验中,对终端节点的计算资源消耗情况进行了测试.实验结果如图 19 和 20 所示.

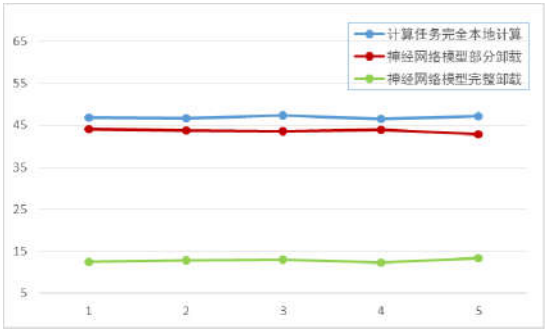


Fig.19 CPU occupancy rate of terminal nodes under different deployment strategies

图 19 不同部署策略下终端节点 CPU 占用率

从图中可以看出,神经网络模型的计算任务对嵌入式端的资源占用较大,神经网络模型部分卸载和完整卸载策略均降低了终端节点的计算资源占用率.在部分卸载策略中,若嵌入式终端设备搭载 GPU 或 NPU 等处理器,应该可以更大程度的降低本地 CPU 占用率,同时降低计算任务的整体响应时间.

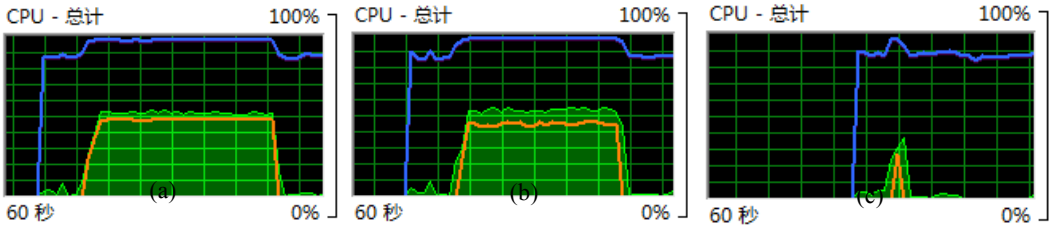


Fig.20 End-node computing resource consumption (orange represents test process) (a) computing task complete local computing (b) partial offloading strategy of neural network model (c) complete offloading strategy of neural network model

图 20 终端节点计算资源消耗(橙色表示测试进程)(a)计算任务完全本地计算(b)神经网络模型部分卸载策略 (c)神经网络模型完整卸载策略

从图 20 可以看出,嵌入式端仅计算 1 层卷积和计算全部神经网络模型,其计算资源占用情况基本相同,仅影响计算资源的占用时长,即仅与任务完成时间相关,其主要原因可能在于实验所采用神经网络模型的计算任务并未进行并行化,不同层卷积运算间的存在固定的先后关联,且不同层卷积运算所占用的计算资源基本相同.

3.3 实验结果分析

实验结果表明,采用任务分割策略可以降低相同计算任务的整体处理时间.降低的时间延迟,主要来源于预处理后传输信息减少所降低的传输延迟.数据预处理与信息量降低本质上并无关联,为降低传输信息量可针对此问题进行特殊设计优化.在目标追踪算法中,在目标位置处裁剪不同尺度的区域作为预处理后的结果进行信息传输,预处理后数据量大小与裁剪区域的大小及数量密切相关.对区域大小和数量进行优化调整,可以使预处理后数据比原始图片的数据量更小.

原始图片数据经过预处理后,其裁剪区域大小仅与追踪目标大小有关,而经过卷积神经网络特征提取后,其大小除了与特征提取区域和神经网络模型设计有关外,还与网络模型参数的存储类型有关,因此,可能出现原数据参数较多,而特征提取后的特征图谱参数较少,但特征提取后占用存储大于原数据的情况.针对该问题,一方面可以针对神经网络模型进行设计改进,借鉴神经网络模型压缩的相关研究,对模型参数进行量化,采用定长存储等方式解决;另一方面,可针对存储后的模型参数文件进行压缩,或在对模型性能影响不大的前提下,删减部分参数.后续将针对该问题进一步研究.

从实验结果可以看出,由于数据类型转换所造成的时间延迟,对任务整体处理时间影响较大.其主要原因在于实现语言及框架限制,该部分时间延迟主要来自深度学习框架下的数据类型向基础数据类型的转换时间,以及将信息打包和解析所造成的时间延迟.终端仅进行预处理操作时.特征数据总的传输开销与传输数据大小、数据转换以及传输框架密切相关.实验中,采用 json 进行数据打包和解析,采用 python requests 库进行 HTTP 连接和数据传输.由于 json 仅支持有限的数据格式,因此,需要进行数据类型转换进行适配.采用 json 对数据进行打包和解析时间较长.该部分时间开销与工程实现方式密切相关,可进一步优化改进.在后续优化中, 将改写其他传输框架,以解决该问题.

针对传输数据的类型和大小,可采用模型参数量化的方式进一步降低数据大小.为避免修改原始网络,在下一步工作中,尝试训练额外的小型神经网络进行中间特征图谱的编码量化.此外,考虑到追踪任务在判断目标位置时,仅与最终响应图中最大值位置有关,因此,若保证最大值位置不变,则其他位置信息损失并不会对模型性能造成影响.尝试通过小型神经网络学习一个近似的特征图谱（或矩阵）,该矩阵具有可分解性质,传输数据为分解后的两个向量.示意图如图所示.此外,考虑是否可以采用传统方法对特征图进行处理,或采用其他数学方式进行近似,以避免添加的网络模型对本地节点产生计算压力.

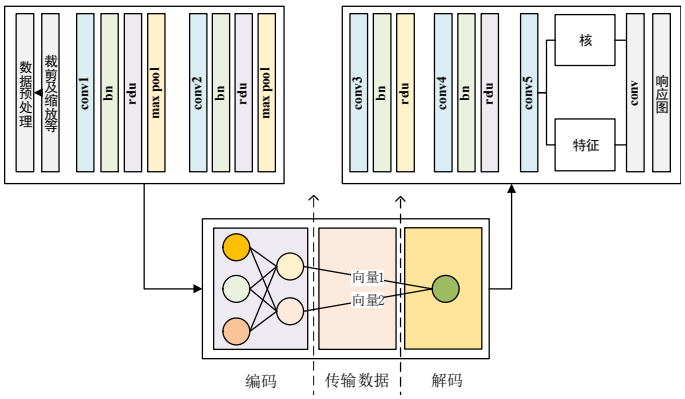


Fig.21 Schematic diagram of feature map parameter quantization scheme

图 21 特征图谱参数量化方案示意图

对于不搭载图形处理单元的终端设备,神经网络模型的计算压力成为整体计算任务的主要瓶颈.对于搭载

轻量级 GPU 或 NPU 的移动终端设备,将神经网络模型继续拆分,将部分计算任务本地计算,从而避开当前网络高延迟时段,仍是值得考虑的部署策略.针对网络传输延迟,对神经网络模型进行优化设计,或根据模型具体结构,从神经网络输出数据量较小的位置进行拆分,如将神经网络模型在升维前后进行拆分等,可能获得更优的处理时间.此外,对于搭载异构处理单元的设备,根据不同处理单元计算性能及优势,训练多个小模型以部署在不同处理单元,最后对多个模型进行集成,或对具体算法模型拆分,分配不同处理单元计算以进一步优化等,在今后的研究工作中,会进行更多探索尝试.

4 结束语

由于目标追踪任务对于实时性的严苛要求,使得移动嵌入式设备难以部署计算压力较大的算法模型.本文结合边缘计算技术,提出一种研究面向边缘计算的目标追踪应用部署策略,通过任务分割策略对计算任务进行划分,并将其合理分配至边缘云端及本地终端节点进行处理,通过协同处理等方式,提高资源的整体利用率,降低任务响应时间.此外,采用响应图重建策略降低信息传输时间,并结合运动检测算法,进一步降低终端节点计算压力,避免不必要计算资源浪费,降低终端节点功耗.最后,通过实验验证该部署策略的有效性,实验结果表明,该部署策略有效降低了相同计算任务处理时间.

结合目前边缘计算和目标追踪的发展现状,可从以下几个方面进一步研究:

(1) 针对特定应用场景,对边缘云架构进行优化设计,将本地计算能力较强的节点上移边缘云层,以降低较差网络环境对任务的影响.此时,任务卸载问题转换为多边缘情况,由此添加针对多边缘场景下的任务卸载策略研究.

(2) 针对目前移动嵌入式设备多搭载多核异构处理器的现状,将本地计算任务进一步细分,以充分利用本地计算资源,如 FPGA、ARM、GPU 等.结合多核异构处理器的调度策略,进一步优化本地资源分配.

(3) 添加多维度约束条件,如功耗,模型性能等,对多约束条件下的资源优化分配问题进行建模,对任务分割策略进行更深入的研究.

(4) 结合模型压缩思想,通过参数量化、模型剪枝等策略,降低终端节点与边缘云之间的信息传输量,进而降低计算任务的响应时间.

References:

[1] Danelljan M, Hager G, Shahbaz Khan F, et al. Convolutional features for correlation filter based visual tracking. Proceedings of the IEEE International Conference on Computer Vision Workshops. 2015: 58-66.

[2] Ma C, Huang J B, Yang X, et al. Hierarchical convolutional features for visual tracking. Proceedings of the IEEE international conference on computer vision. 2015: 3074-3082. DOI: 10.1109/ICCV.2015.352

[3] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[4] Tao R, Gavves E, Smeulders A W M. Siamese instance search for tracking. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 1420-1429. DOI: 10.1109/CVPR.2016.158

[5] Bertinetto L, Valmadre J, Henriques J F, et al. Fully-convolutional siamese networks for object tracking. European conference on computer vision. Springer, Cham, 2016: 850-865.

[6] Li B, Yan J, Wu W, et al. High performance visual tracking with siamese region proposal network. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 8971-8980. DOI: 10.1109/CVPR.2018.00935

[7] Li B, Wu W, Wang Q, et al. Siamrpn++: Evolution of siamese visual tracking with very deep networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 4282-4291.

[8] Zhang Z, Peng H. Deeper and wider siamese networks for real-time visual tracking. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 4591-4600.

[9] Wang Q, Zhang L, Bertinetto L, et al. Fast online object tracking and segmentation: A unifying approach. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 1328-1338.

[10] Shi W, Fellow, IEEE, et al. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, 2016, 3(5):637-646.

[11] Hu W, Gao Y, Ha K, et al. Quantifying the impact of edge computing on mobile applications. Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems. ACM, 2016: 5.

[12] Garg S, Singh A, Kaur K, et al. Edge Computing-Based Security Framework for Big Data Analytics in VANETs. IEEE Network, 2019, 33(2): 72-81. DOI: 10.1109/MNET.2019.1800239

[13] Sheng Z, Pfersich S, Eldridge A, et al. Wireless acoustic sensor networks and edge computing for rapid acoustic monitoring. IEEE/CAA Journal of Automatica Sinica, 2019, 6(1): 64-74. DOI: 10.1109/JAS.2019.1911324

[14] Lai C F , Chien W C , Yang L T , et al. LSTM and Edge Computing for Big Data Feature Recognition of Industrial Electrical Equipment. IEEE Transactions on Industrial Informatics, 2019, PP(99):1-1. DOI: 10.1109/TII.2019.2892818

[15] Muhammad G , Alhamid M F , Alsulaiman M , et al. Edge Computing with Cloud for Voice Disorder Assessment and Treatment[J]. IEEE Communications Magazine, 2018, 56(4):60-65. DOI: 10.1109/MCOM.2018.1700790

[16] Zishu LI, Renchao XIE, Li SUN, Tao HUANG. A survey of mobile edge computing. Telecommunications Science, 2018, 34(1): 87-101. (in Chinese with English abstract). DOI: 10.1109/JIOT.2016.2579198

[17] Tong L, Li Y, Gao W. A hierarchical edge cloud architecture for mobile computing. IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, 2016: 1-9. DOI: 10.1109/INFOCOM.2016.7524340

[18] Yao C, Wang X, Zheng Z, et al. EdgeFlow: Open-Source Multi-layer Data Flow Processing in Edge Computing for 5G and Beyond. IEEE Network, 2018:1-8. DOI: 10.1109/MNET.2018.1800001

[19] Chia-Wei T, Fan-Hsun T, Yao-Tsung Y, et al. Task Scheduling for Edge Computing with Agile VNFs On-Demand Service Model toward 5G and Beyond. Wireless Communications and Mobile Computing, 2018, 2018:1-13.

[20] Sun Y, Guo X, Song J, et al. Adaptive Learning-Based Task Offloading for Vehicular Edge Computing Systems. IEEE Transactions on Vehicular Technology, 2019. DOI: 10.1109/TVT.2019.2895593

[21] Zhang Y, Chen X, Chen Y, et al. Cost efficient scheduling for delay-sensitive tasks in edge computing system. IEEE International Conference on Services Computing. IEEE, 2018: 73-80. DOI: 10.1109/SCC.2018.00017

[22] Fan Q, Ansari N. Application Aware Workload Allocation for Edge Computing-Based IoT. IEEE Internet of Things Journal, 2018, 5(3):2146-2153. DOI: 10.1109/JIOT.2018.2826006

[23] Du W, Lei T, He Q, et al. Service Capacity Enhanced Task Offloading and Resource Allocation in Multi-Server Edge Computing Environment. arXiv preprint arXiv:1903.04709, 2019.

[24] Peiyuan G, Xiaoheng D, Yajun L, et al. Analysis of Multiple Clients' Behaviors In Edge Computing Environment. IEEE Transactions on Vehicular Technology, 2018:1-1. DOI: 10.1109/TVT.2018.2850917

[25] Lin B, Zhu F, Zhang J, et al. A Time-driven Data Placement Strategy for a Scientific Workflow Combining Edge Computing and Cloud Computing. IEEE Transactions on Industrial Informatics, 2019. DOI: 10.1109/TII.2019.2905659

[26] Ren Y, Nitu V, Liu G, et al. Efficient, Dynamic Multi-tenant Edge Computation in EdgeOS. arXiv preprint arXiv:1901.01222, 2019.

[27] Henriques J F, Caseiro R, Martins P, et al. High-speed tracking with kernelized correlation filters. IEEE transactions on pattern analysis and machine intelligence, 2014, 37(3): 583-596. DOI: 10.1109/TPAMI.2014.2345390

[28] Henriques J F, Caseiro R, Martins P, et al. Exploiting the circulant structure of tracking-by-detection with kernels. European conference on computer vision. Springer, Berlin, Heidelberg, 2012: 702-715.

[29] Wu Y, Lim J, Yang M H. Object Tracking Benchmark. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(9):1834-1848. DOI: 10.1109/TPAMI.2014.2388226

[30] Huang L, Zhao X, Huang K. GOT-10k: A Large High-Diversity Benchmark for Generic Object Tracking in the Wild. arXiv preprint arXiv:1810.11981, 2018.

[31] Zhang XQ. Research on target tracking algorithms for edge computing [M.S. Thesis]. Harbin Institute of Technology, 2019. (in Chinese with English abstract)

附中文参考文献:

[16] 李子姝, 谢人超, 孙礼, 黄韬. 移动边缘计算综述. 电信科学, 2018, 34(1):87-101.

[31] 张宪琦. 面向边缘计算的目标追踪算法研究[D]. 哈尔滨工业大学, 2019.