分类号：TP311
密　级：

单位代码：10422
学　　号：201613447


山东大学
SHANDONG UNIVERSITY

# 硕士学位论文

Thesis for Master Degree

论文题目：

基于嵌入式异构 GPU 平台的实时目标检测系统设计

**DESIGN OF REAL-TIME OBJECT DETECTION SYSTEM BASED ON EMBEDDED HETEROGENEOUS GPU PLATFORM**

作 者 姓 名　柳佳园
培 养 单 位　计算机科学与技术学院
专 业 名 称　计算机科学与技术
指 导 教 师　黎峰　副教授
合 作 导 师

2019 年 05 月 20 日

分类号：TP311　　单位代码：10422

密　级：　　学　　号：201613447

山东大学

# 硕士学位论文

论文题目：基于嵌入式异构 GPU 平台的实时目标检测系统设计

**DESIGN OF REAL-TIME OBJECT DETECTION SYSTEM BASED ON EMBEDDED HETEROGENEOUS GPU PLATFORM**

作 者 姓 名　柳佳园

学 院 名 称　计算机科学与技术学院

专业学位名称　计算机科学与技术

指 导 教 师　黎峰 副教授

合 作 导 师

2019 年 5 月 20 日

# 原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名：柳佳园　　日　期：2019-5-20

# 关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权山东大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名：柳佳园　导师签名：　日　期：2019-5-20

# 目 录

# CONTENTS

# 摘 要

目标检测在信息技术以及人工智能领域有着广泛的应用，包括机器人视觉、自动驾驶、无人机智能监控等。各种各样的应用场景对在嵌入式设备上实现高精度且高能效的实时目标检测提出了迫切的需求。

目标检测是计算机视觉领域最具挑战的问题之一，它不仅要对图像中的物体进行分类，还要对其进行定位。传统的解决方案一般采用滑动窗口的方法，利用训练好的分类器对所有的可能窗口进行判断分类，具有非常高的计算复杂性，且准确率低，有着明显的局限性。而随着深度学习的不断发展和应用，许多研究将卷积神经网络引入目标检测算法的实现并不断进行改进，使得目标检测的性能越来越高，特别是 SSD、YOLO 等端到端的目标检测模型的提出，使得检测的速度取得了新的突破。

虽然目前的目标检测技术已经具备了非常高的检测精度和速度，但是当应用到计算和内存资源都有限的嵌入式平台上时，性能大幅下降，达不到实时性标准。因此，实现嵌入式平台上的实时目标检测仍然是一个很大的挑战。

本文的研究选择基于嵌入式异构 GPU 平台 NVIDIA JETSON TX2 进行，在目标检测技术学习总结的基础上，将性能较高的几个模型部署到 TX2 上进行对比分析，经过权衡准确度和速度两个重要指标，选择将 YOLOv2 模型进行优化研究，达到嵌入式平台上实时目标检测系统严格的检测准确度和实时性要求。优化方案主要有以下几个方面：第一、对 YOLOv2 网络层次结构和参数进行调整，通过多次实验探索，确定了最适合系统的网络超参数，通过减少卷积核以及网络层数来减少网络运算量；第二、使用半精度浮点数，将网络中的 32 位浮点数转换为 16 位，从而减少了数据传输量和运算量，通过该方法，模型的准确度略有下降，但速度得到了明显的提升；第三、分析代码在 CPU 和 GPU 上的运行耗时情况，针对平台特点，进行多线程的流水线设计，获得了 4fps 的速度提升。这三种优化方案主要根据网络本身的结构和实际平台的特点设计实现的，针对不同的数据集具有普适性。另外，根据对运动目标进行实时检测从而实现追踪的需求，考虑到将 YOLOv2 与追踪算法结合，经过实验发现，在对视频中的运动目标检

测时，将 GOTURN 目标追踪算法与优化后的 YOLOv2 的结合调度使得系统达到更高的系统性能。

**关键词：** 目标检测；嵌入式 GPU；卷积神经网络；

# Abstract

Object detection is widely used in the field of information technology and artificial intelligence, including robot vision, autonomous driving, intelligent monitoring on unmanned aerial vehicles(UAV) and so on. A variety of application scenarios put forward an urgent demand for real-time object detection with high precision and high energy efficiency on embedded devices.

Object detection is one of the most challenging problems in the field of computer vision, it not only needs to classify objects in the image, but also locate them. Traditional solutions usually adopt the sliding window approach and use the trained classifier to judge all the possible windows, which leads to significant limitations in terms of not only high computational complexity but also high detection error rate. With the continuous development of deep learning, the realization convolutional neural network(CNN)-based methods achieves increasingly high performance. Particularly, some end-to-end object detection models such as SSD and YOLO have made a new breakthrough in the speed of object detection.

Although the current object detection technology has a very high detection accuracy and speed, when applied to embedded platforms with limited computing and memory resources, the performance of these technologies is greatly reduced and can not meet the real-time standard. Therefore, it is still a great challenge to realize real-time object detection on embedded platform.

This paper chooses NVIDIA JETSON TX2, an embedded heterogeneous GPU platform, to carry out the research. On the basis of learning and summarizing the technology of object detection, several models with higher performance are deployed to TX2 for comparative analysis. After weighing the two important indicators of accuracy and speed, the YOLOv2 model is chosen to optimize and research, so as to achieve the real-time object detection system on embedded platform. The optimization scheme mainly includes the following aspects: First, modify the

YOLOv2 network structure and parameters to accelerate the network. Second, adopt half-precision floating-point numbers to reduce the amount of computation in the network. With this method, the speed is significantly improved at a slightly cost of the accuracy. Third, analyze the time consumption on the CPU and GPU during the detection and design the multi-threaded pipeline that obtains the speed improvement of 4FPS. The above three optimization schemes are mainly designed and implemented according to the structure of the network itself and the characteristics of the actual platform, and are applicable to different data sets. In addition, according to the requirement of real-time detection of moving objects to achieve tracking,we combine optimized YOLOv2 with the tracking algorithm. It is found through experiments that that when detecting moving objects in video, the combination of GOTURN target tracking algorithm and optimized YOLOv2 makes the system achieve higher system performance.

**Keywords:** Object detection; Embedded GPU; Convolutional neural network;

# 第 1 章 绪论

## 1.1 研究背景及意义

目标检测作为视觉识别问题的基本任务之一，一直是世界范围内的研究热点[1]。随着 GPU 和 CUDA 的发展，使得计算机的计算能力得到爆发性的提高，GPU 在通用计算方面的使用，大大提升了深度学习网络特别是卷积神经网络的训练和测试速度[2]，成就了深度学习在语音识别、计算机视觉等领域的应用中突破性进展，从而使得目标检测技术的发展也突飞猛进。

2005 年到 2012 年举办的 Pascal VOC(Visual Object Classes)比赛见证了目标检测技术从传统技术到基于深度学习的目标检测技术的发展过程。其中同时实现了图片分类、定位以及物体检测的 Overfeat[3]算法是目标检测引入卷积神经网络（Convolutional Neural Networks, CNN）的转折点。之后的目标检测网络也持续的发展和更新，通过利用网络结构的调整和图像预处理技术的提升等实现目标检测网络的不断改进。

目标检测在信息技术以及人工智能等各个领域都有着广泛的应用空间，例如自动驾驶、安保系统、无人机及智能监控等。随着目标检测在各领域的推广和使用，越来越多的应用场景对在嵌入式设备上实现高精度且高能效的实时目标检测提出了迫切的需求，而大部分嵌入式设备包括智能手机、嵌入式视频监控以及无人机等终端设备由于计算和存储资源有限，大大增加了移动端目标检测的高准确性以及实时性的实现难度。而嵌入式 GPU 的发展使得嵌入式设备的计算能力大大提升。目标检测技术的不断发展以及嵌入式设备性能的提高，为在嵌入式 GPU 平台上实现高精度且高能效的实时目标检测奠定了基础。

学术界和产业界对基于深度学习的目标检测技术的发展一直非常关注，并对其开展了众多的研究和应用，众多的研究表明基于深度学习的目标检测技术已经具备了非常高的检测精度和速度[4]。但是大部分研究和测试都是基于高性能的实验平台进行的，当应用到计算资源和内存资源都有限的嵌入式平台上时，其整体

性能会大幅下降，远远达不到实际应用标准。因此基于嵌入式平台的实时目标检测系统设计研究还需进一步探索，且该研究对于目标检测技术在各个领域的应用和推广具有非常重要的意义。

## 1.2 研究现状分析

随着社会生产技术的发展和提高，多样化的需求环境对目标检测的速度和准确度要求越来越高。而由于光照、遮挡等原因，传统的检测技术在复杂多样的工业环境中难以满足实际应用的需求标准。

2014 年，受到图像分类引入 CNN 后取得了突破性成果的启发，Ross B. Girshick 第一个探索将 CNN 用于目标检测，提出的 R-CNN 目标检测算法[12]，该研究打开了目标检测领域的新世界，吸引了各界陆续开展对基于深度学习目标检测算法的研究。随后，He 等人针对 R-cnn 提出了改进的算法，将空间金字塔池化层引入 CNN 架构，提出了 SPP-net 检测算法[15]，该算法中神经网络支持任意的输入尺寸，其检测速度可以达到 R-CNN 的几十倍甚至 100 倍。2015 年，Girshick 提出了 Fast R-CNN[13]，不需要存储训练和测试过程中的中间值，将检测速度提升了十倍。同年，Ren 等人提出的 Faster R-CNN 算法[14]，并提出一个基于 CNN 的候选区域生成网络 RPN 取缔了之前 region proposal 阶段采用的 selective search（选择性搜索）算法，并且 RPN 与进一步减少了时间开销。在此基础上，后续的研究不断进行改进，开发了 RFCN 和 mask R-CNN 等框架。

上述框架算法在目标检测过程中首先要生成候选区域，在此基础上进行分类判断和位置计算，被称为 two-stage 目标检测算法。虽然 two-stage 目标检测算法的研究已经取得了非常优秀的进展，但其计算比较复杂，难以适用存储和计算有限的嵌入式设备。研究者已经开始了对 one-stage 的目标检测策略。Sermanet 等人提出的 overfeat 网络比同期的 R-CNN 有明显的速度优势，但是准确度较低。2015 年，Redmon 等人提出的 YOLO 框架[16]，速度可达 45fps。之后提出的 SSD[19] 算法针对 YOLO 定位准确性低的问题做出了进一步改善。2017 年 YOLO 的研究团队推出了 YOLOv2 版本[17]，在准确度和速度上均取得了较大的突破。至今，该团队仍保持对 YOLO 的研究学习，且于 2018 年提出了 YOLOv3[18]。

至今，基于深度学习的目标检测发展的研究仍在持续，且成果斐然。面对实际应用的迫切需求，也有大量的研究学者开始了基于嵌入式开发平台的目标检测研究。而嵌入式系统除了计算和存储资源有限外，能耗也是一个重点问题。而FPGA（现场可编程门阵列）具有实现高性能和相对低功耗的潜力，因此有关基于FPGA进行CNN以及基于CNN计算机视觉任务的优化设计。文献[26]研究了嵌入式FPGA平台上优化基于CNN的目标检测算法的新方法，给出了基于FPGA的CNN设计模型以及针对FPGA的优化流程。文献[27]中，作者在嵌入式FPGA平台上成功部署VGGNet，并采用了数据量化和系数矩阵分解等多种优化技术进行优化。

而目前嵌入式平台上的大规模的复杂网络部署还是更多的采用GPU实现。2016年NVIDIA公司设计了专门针对深度学习神经网络的嵌入式开发平台Jetson TX1，第二年又推出了第二代产品Jetson TX2，更是获得了比TX1翻倍的性能优势，以及2018年出的性能更高的Jetson Xavier为基于深度学习神经网络的研究提供了有利条件。近两年，涌现了大量基于上述NVIDIA嵌入式开发平台的目标检测研究。Mao等人基于TX1平台对嵌入式系统的实时目标检测系统进行研究[28]，他们选用fast R-CNN方法并对其进行修改以适用嵌入式平台，实现速度和检测精度之间的平衡，其解决方案获得了低功耗图像识别挑战赛（LPIRC）获得了一等奖，虽然其能耗和准确度方面有一定的优势，但其速度仅为1.85fps。Tijtgat N等人基于TX2平台设计了用于无人机的实时目标检测预警系统[29]，通过对几个尖端物体检测算法的对比，选用YOLOv2模型，但该文章只对网络的配置进行了探索，没有进一步优化设计，使用YOLOv2时的检测速度最高不到10fps。文献[30]研究了深度神经网络（DNN）系统在自主机器人导航中的目标检测和距离估计的应用，选择SSD目标检测模型，底层网络为MobileNet。

因此可以看出，基于嵌入式平台和移动端的目标检测研究已经逐渐成为当前领域的研究热点，且基于嵌入式设备上的目标检测技术的应用，仍然具有一定的优化空间。

## 1.3 本文主要工作

嵌入式 GPU 大大提高了嵌入式设备的计算能力，为实现实时的目标检测系统提供了可能。随着目标检测在航空拍摄以及实时监控等多种场景的广泛应用，提出了对拍摄画面中的对象实现目标检测的需求，通过对视频每一帧画面进行快速地目标检测从而实现对目标的实时定位和识别。

为解决上述需求，本文对如何实现嵌入式平台上的实时目标检测展开研究。为了充分利用 GPU 的计算能力优势，选择在低功耗的嵌入式异构 GPU 平台 NVIDIA Jetson TX2 上对实时目标检测系统进行设计。由于 TX2 的功率低，所以在设计时，通过尽可能的减少系统工作时的计算量来控制系统的能耗，同时可以提升系统的速度。而计算量的减少往往伴随着检测精度的下降，如何在保证系统检测准确度的前提下提升检测的速度，是本文要解决的核心问题。因此本文的主要工作如下：

(1) 目标检测模型实验对比。为了选择适合我们的嵌入式平台上的目标检测模型。通过对现在前沿的几个目标检测模型部署到 TX2 上，对比分析了不同模型基于 TX2 实现的检测速度和准确度，在权衡两个指标后选用基于 darknet 框架实现的 YOLOv2 模型进行进一步的优化研究。

(2) 调整网络结构和参数。在充分理解 YOLOv2 模型的结构和原理的基础上，对 YOLOv2 的结构进行改进。首先通过参数的调整，从而使原始的网络达到最好的准确度。然后，通过减少网络层数，减少 YOLOv2 卷积过程的计算量，来获得更高的检测速度。调整后的网络检测速度达到 16fps 左右，平均 IoU 为 0.69 左右。

(3) 基于 TX2 体系结构的优化研究。TX2 支持半精度也就是 16 位浮点数的计算，且其运算速度比 32 位浮点数的计算速度快。因此采用半精度浮点数提升速度。将图像预处理后将 32 位浮点数转换位 16 位，且将网络中涉及的浮点数计算都转换为半精度浮点数计算，这样不但减少了运算量，也减少了 CPU 到 GPU 的数据拷贝量。另外，TX2 上有多核 CPU，因此通过多线程 CPU-GPU 流水的设计来最大化地利用 GPU 的计算能力。通过统计分析模型在进行检测时代码在

CPU 和 GPU 的运行时间分配情况，设计使用多线程 CPU、GPU 流水来提升系统的效率。通过优化，网络的检测速度达到 24fps 左右，

(4) 基于运动目标的检测优化研究。目标跟踪算法由于图像目标遮挡等原因，其准确度往往较基于深度学习的目标检测低，但其速度非常高。因此采用 YOLOv2 与 GOTURN 网络协同调度的方式，即 YOLOv2 每识别一张图片，GOTURN 识别五张图片的组合方式，在提高系统检测速度的同时，保证准确度的合理性。另外，根据平滑运动的物体运动轨迹的规律性设计实现位置预测算法的设计。针对对目标的实时检测从而实现跟踪的需求。在总结目标的运动规律基础上，设计了位置预测算法，将其与 YOLOv2 网络相结合，最终将系统的速度提升至 30.4fps。

## 1.4 本文组织结构

本文主要研究目标是基于嵌入式异构 GPU 平台 TX2 实现实时可靠的目标检测系统。首先了解相关技术的发展，在分析课题背景和研究意义的基础上，明确研究目标。根据目标检测技术的发展情况，将性能较高的几个前沿的检测模型部署到 TX2 平台进行测试，充分考量各目标检测模型的检测速度和准确度等指标，最终确认选取 YOLOv2 检测模型。最后，根据系统目标，制定了一系列的优化方案，并在实际平台上进行改进，最终达到理想的效果。

本文的章节安排如下：

第一章，绪论部分介绍了目标检测技术的发展背景，分析了本文课题研究的理论意义以及实际应用价值，对本文的主要工作进行总结。

第二章，介绍了相关技术背景。首先介绍了本文的硬件平台 TX2，然后总结了传统目标检测技术到先进的基于深度学习的目标检测技术的一步步改进，分析了各种基于深度学习的目标检测模型的基本原理和优缺点。

第三章，在了解目标检测技术的基础上进行了实验，权衡性能指标，选取适合嵌入式开发平台 TX2 的目标检测模型。根据实验数据选取基础网络为 darknet19 的 YOLOv2 作为本文系统研究的目标检测模型。

第四章，根据网络结构和 TX2 平台的特点，设计实现了一系列网络优化方法，将基于 TX2 实现的目标检测速度提升至 23fps 左右。

第五章，根据项目实际应用需求，将优化过的目标检测模型与目标追踪相结合，并通过实验证明了方案的可行性。最终通过优化在实验平台上实现了高精度的实时性目标检测。

最后是结论，对本文的研究进行概括总结。

# 第 2 章 相关技术背景

本文目标检测研究基于 NVIDIA JETSON TX2 平台实现，该平台体积小巧，集成度高，性能强大，是比较领先的低功耗异构嵌入式平台，适用于无人机、机器人等智能终端设备。通过基于 TX2 上不同目标检测模型的对比实验分析，在比较了各模型的速度和准确度的基础上，选用 YOLOv2 作为本文的研究模型。

## 2.1 硬件分析

TX2 拥有 6 核 ARMv8 64 位 CPU 复合体和一个 256 核 NIVIDIA Pascal 架构 GPU，其中 CPU 复合体包括双核 Denver2 处理器和四核 ARM Cortex-A57 组成，除此之外，还有 8GB LPDDR4 内存和 128 位接口，非常适合低能耗和高计算性能的应用场景，因此选择该平台来设计实现实时目标检测系统，并基于该平台 GPU 结构及特性设计优化方案。

GPU（graphic processing unit，图形处理单元），其最初的设计主要为了满足图像运算中复杂的几何及数学计算要求，GPU 的出现将 CPU 从复杂的图形处理的任务中解放出来，从而大大提升了计算机的整体性能。随着 GPU 通用计算技术的发展，GPU 不再局限于图形处理，其在浮点计算、并行计算等方面表现出非常强悍的计算能力，使其在深度学习等领域有了广泛的应用。在 GPU 在通用计算方面的标准有 OpenCL、CUDA 等。NVIDIA 公司可以说是最大的 GPU 芯片生产商，CUDA 就是该公司提供的编程模型

CPU 中将大部分空间留给了控制单元和缓存空间，而 GPU 设计者则将更多的芯片空间用留给 ALU，这也是 GPU 计算能力强悍的原因。CPU 要处理各种不同的数据类型，而且需要同时支持并行及串行操作还有复杂的逻辑判断和流控制等，因此要求 CPU 具有很强的通用性，使得 CPU 控制部分比较复杂，计算单元的比重就被减少了。而 GPU 的任务则是类型非常统一的大规模数据的计算，因此 GPU 的芯片比较简单，计算单元比较多而不需要复杂的控制结构。

一般一个 GPU 有许多流多核处理器（SM）。每个 SM 由多个流处理器（SP）、

共享内存、寄存器等组成。其中 SP 是 GPU 最基本的处理单元，GPU 的并行计算时就是很多个 SP 在同时计算。每个 SM 由多个流处理器（Streaming Processor，SP）、共享内存、寄存器等组成。其中 SP 是 GPU 最基本的处理单元，与线程 thread 对应，各种指令以及任务都是在 SP 上进行的，GPU 的并行计算时就是很多个 SP 在同时计算。一个 GPU 应用程序会发起一个或多个核（kernel）函数到 GPU 上，其中 kernel 函数指的就是在 GPU 上执行的函数，每个 kernel 函数会分派一个或多个 CTA（Co-operative Thread Arrays）给每个 SM，一个 CTA 上的每 32 个线程组成一个 warp，warp 就是线程束，是线程调度的基本单位，一个 warp 内的线程执行的指令操作是相同的而所拥有的数据资源是不同的，每个线程都有对应的寄存器内存和 local memory。而 warp 同时也是一个硬件相关的概念，一般一个 SM 内的 SP 会分成多个 warp。

CPU+GPU 异构解决了 CPU 无法快速处理大量数据计算的缺陷。在 CPU 和 GPU 异构的平台系统中，通常情况下两者都拥有独立的外部存储[35]。其中,CPU 的外部存储器称为主存，GPU 的外部存储器称为显存。CPU 和 GPU 之间由北桥通过 AGP 或 PCI-E 总线连接，其中 CPU 主要负责处理逻辑性比较强的任务，而 GPU 则用来处理高密度的浮点计算。如图 2-1 所示，数据通过北桥由主存传入到 GPU 的显存中，GPU 完成对数据的处理计算后，将结果传出到 CPU 的主存。

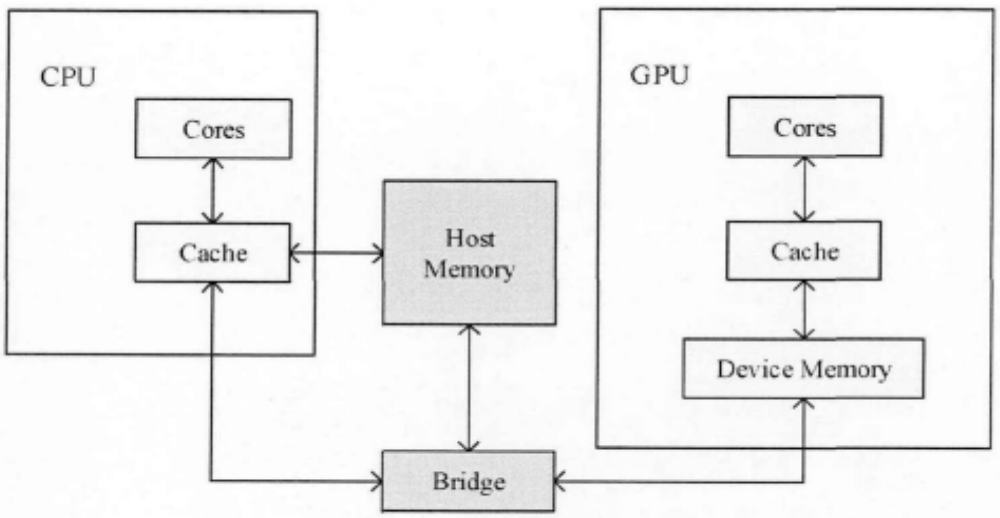


图 2-1 CPU/GPU 异构结构示意图

## 2.2 CUDA 编程模型

CUDA（Compute Unified Device Architecture，统一计算设备架构）[36]是英伟达推出的 GPU 编程模型。CUDA 程序文件以.cu 为后缀，采用开发人员比较熟悉的 c 语言进行开发。在 CUDA 程序文件中，CPU 一般被称作 host，GPU 称作 device，host 只有一个，而 device 可以有多个即一个 host 可以对应一个或多个 device。完整的 CUDA 程序由两部分组成，一部分是主机端代码，在 CPU 内执行，一般负责任务的组织和发送；另一部分是 kernel 函数，在 GPU 执行，负责并行计算任务。

为了便于对 kernel 函数的理解，要了解 thread，block 以及 grid 等几个概念，grid 是分配给每个 kernel 函数的总资源，由多个 block 组成，grid 之间共享 gpu 的 global memory；而 block 即线程块，是分配到 SM 的基本单位，同一个 block 不能拆分到不同的 SM 上，只由一个 SM 调度，block 由线程组成，block 内的线程共享 shared memory 以及 L1 cache；而每个线程都有各自的寄存器、local memory 等私有资源。CUDA 是典型单指令多线程 SIMT 架构，所谓 SIMT 就是指同一个 warp 内的线程执行的指令是相同的，但数据资源是不同的。

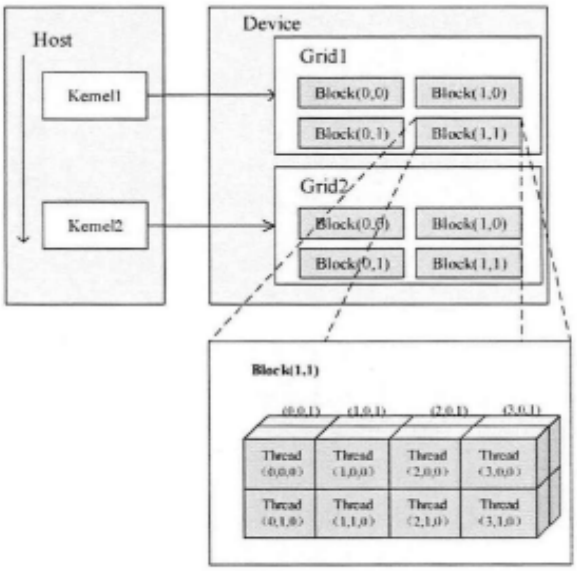


图 2-2 CUDA 的线程组织结构

在英伟达提供的 CUDA 5.0 之后的版本，将 CUDA、驱动程序以及 SDK 统

一集成到 CUDA Toolkit 安装包中，该安装包还提供复杂编译 CUDA 代码的 NVCC 编译器、CUDA 库函数等各种 CUDA 编程所需要的基础软件环境。本文系统的实验是基于 Linux 系统进行研究的，实验平台采用的是 Ubuntu16.04，根据该环境选择的 CUDA-8.0 版本。

通过对 CUDA 编程模型的介绍，可以看出 block 和 thread 是 CUDA 中的两个并行模式的逻辑层，在执行过程，block 对应流多核处理器 SM，thread 对应 SP 流处理器。因此，对这两个逻辑层上程序并行性的提高，是提高 CUDA 整体执行性能的关键，同时也是优化 CUDA 程序的关键。

## 2.3 目标检测技术发展

目标检测与目标分类不同，目标分类只需要预测图片中的对象类别，而目标检测在分类的基础上还要找到该目标在图片中的位置。对于我们自身来说目标检测是非常简单的任务，我们可以根据自身经验很容易且迅速地识别出图片中的不同物体。然而，对计算机来说，图片是数字化的，它看到的图片是一些数组，很难直接判断出图片中物体及其位置。而图片中的目标位置的不确定性、目标形态以及图片背景的多样性使得目标检测对于计算机来说并不是一个简单的任务。

Haar 特征+Adaboost 算法、HOG 特征+SVM 算法、DPM 算法[10]是以往的应用和研究中常用的三种传统目标检测技术组合。传统目标检测的方法在进行检测时，整个过程通常分为三个步骤：

(1) 区域选择：利用尺寸大小不同的滑动窗口遍历整张图，在图片中圈出一部分作为候选区域；这是一种穷举策略，因此会有非常高的时间复杂性，产生大量的冗余窗口，对后续的特征提取和分类的速度和性能产生严重的影响。

(2) 特征提取：这个阶段的主要任务是在前一步获取的候选区域的基础上，提取视觉特征。常见的特征有用于人脸检测的 Harr 特征[6]，用于行人检测及普通目标检测的 HOG 特征[7]以及 SIFT 特征[8]，。由于目标的形态、光照以及图片背景等因素的变化多样性，使得设计一个具有鲁棒性的特征并不简单，然而所提取特征的优劣对后续分类的准确性具有直接的影响。

(3) 分类器分类：最后一个阶段的任务则是利用 SVM、Adaboost 等分类器对提取的特征进行分类，最终给出图片中所识别的目标。

从上节传统目标检测三个步骤及算法介绍可以看出传统目标检测技术的通病，主要存在两个方面：一个是利用滑动窗口选取候选区域的策略没有针对性，会导致非常高的时间复杂度和大量窗口冗余；二是手工设计的特征对于目标以及光照等因素的多样性变化没有很好的鲁棒性。

而基于深度学习的目标检测算法中，region proposal（区域提名）的提出很好的解决了上述采用滑动窗口出现的问题。基于传统区域选择算法 region proposal 方法有 selective search[20]和 edge boxes[21]，上述算法利用图片中的颜色、边缘、纹理等信息，预先找出目标在图片中可能出现的位置也就是候选区域，并且可以保证在选取窗口数量大大减少的的情况下而保持较高的召回率。另外一种 region proposal 方式是采用 RPN（Region Proposal Network，区域选择网络），该网络基于卷积神经网络实现。

对于特征提取和分类方面，在 2012 年 ImageNet 大规模视觉识别挑战赛（ILSVRC）[44]上，Geoffrey Hinton 教授带领学生采用卷积神经网络取得了比赛的第一名，将 ILSVRC 分类任务的 Top-5 错误率降低到 15.3%，而取得第二名队伍使用了传统算法，其 Top-5 错误率高达 26.2%。2014 年，R-CNN 框架问世，该框架采用 region proposal 技术和卷积神经网络，替代了传统目标检测所使用的滑动窗口和手工设计特征，该框架取得了目标检测领域的巨大突破，也引起了研究者对基于深度学习目标检测技术的研究热潮。

目前采用的主流的目标检测算法基本上都是基于深度学习模型实现的，采用卷积神经网络来进行特征的提取。基于深度学习的目标检测算法主要分为 two-stage 检测算法和 one-stage 检测算法[4]。

### 2.3.1 CNN 卷积神经网络

基于深度学习的目标检测算法采用很深层次的卷积神经网络提取出图像中更能反应目标本质的高维特征，然后利用这些特征对图像中的目标进行分类和定位。卷积神经网络主要层级结构有数据输入层、卷积层、ReLU 激励层以及全连

接层。比较常见的卷积网络结构有 AlexNet[37]、VGGNet[38]、GoogleNet[39]以及 ResNet[40]等。

数据输入层即负责对输入的图像数据进行归一化处理。

卷积层通过卷积运算，初步提取图像特征。在卷积层中，利用卷积核（也称过滤器），基于滑动窗口的思想，对输入图像的数据进行卷积运算。这里涉及到 3 个概念，分别是深度、步长及填充值 padding。深度一般与图片的 3 个 RGB 颜色通道对应，值为 3；步长指的是窗口每次滑动的长度；填充值指的是在图片矩阵周围填充 0，填充的圈数与填充值对应，使得窗口能够正好滑到图片边界。

如图 2-3 所示是是卷积运算的例子中，输入为 7×7×3 的三维张量，由原输入周围填充一圈 0 得到的结果，使用两个卷积核 W0 和 W1，即图中红色的 3×3×3 张量，每个颜色通道上都使用一个 3×3 的滑动窗口，步长设为 2，将窗口里的值与卷积核里的值对应相乘，这样每个颜色通道上都能得到一个乘积值，将这三个值相加得到最后的输出，由于使用了两个卷积核，则卷积的结果为 3×3×2 的张量。

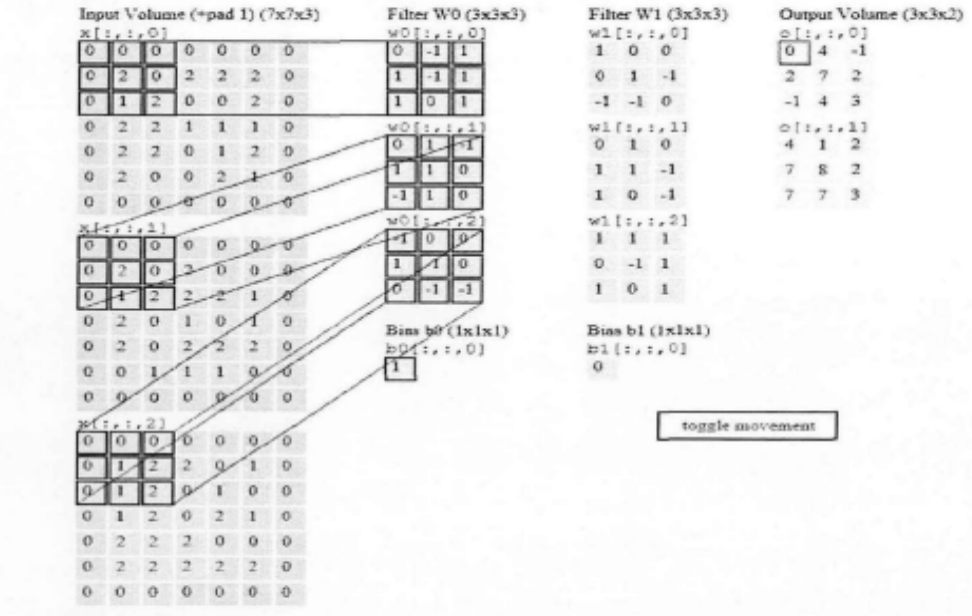


图 2-3 卷积运算示例

在激励层，利用激励函数对卷积层的输出特征结果做非线性映射。通常采用 ReLU 激励函数把输出结果中负值对应的位置改为 0。

池化层，也称下采样层。一般位于连续的卷积层之间，其主要目的是对卷积层输出的特征向量结果和参数压缩降维，同时有效减少过拟合现象。图 2-4 展示了最常用的池化操作——最大池化和平均池化。最大池化就是选取区域内最大值代表该区域，平均池化则是计算区域内数值的平均值来代表该区域。池化层提取到的是主要特征。

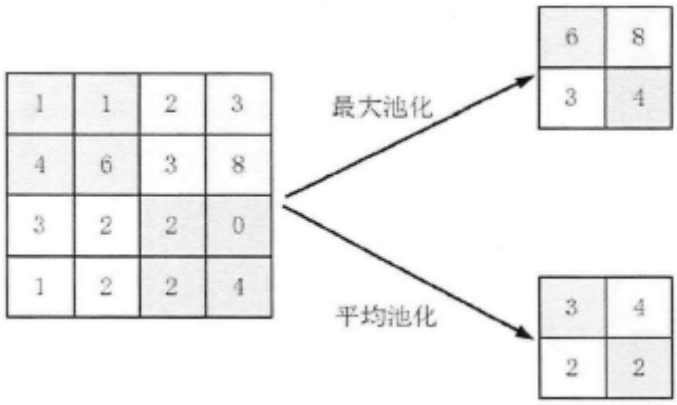


图 2-4 最大池化层的和平均池化示例图

全连接层一般位于网络的最后，负责综合汇总各部分特征，将经过卷积和池化提取的局部特征组装成完整的图，并给出各种分类类别对应的概率，为之后的分类提供依据。

### 2.3.2 Two-stage 目标检测算法

Two-stage 检测算法，就是基于区域提名的目标检测方法，这类算法将检测过程大体分为两个阶段。首先是生成候选区域，对候选区域利用卷积神经网络提取特征，然后使用相应的分类器进行分类和定位。

1. R-CNN

R-CNN，它可以说是第一个工业级的基于深度学习的目标检测技术，R-CNN 网络检测目标的主要步骤如下：

(1) 输入测试图片；

(2) 选取候选区域：采用选择性搜索算法（Selective Search）在该图像上提

取 2000 个候选区域；

(3) 特征提取，首先要统一候选区域大小，由于上步所取出的候选区域大小各自不同，而 CNN 中输入到全连接层的数据长度要求是固定的，所以需要对候选区域框进行缩放，处理成统一的大小后输入到 CNN，将 CNN 最后一层全相连层的输出作为特征；

(4) 分类与回归：采用 SVM 分类器对每个候选区域框提取到的 CNN 特征进行判断。至此便可以得到目标所在的候选区域，再采用回归的方法对微调目标边界框的位置和大小，输出最终结果。

虽然在选取候选区域时，R-CNN 改变了之前一直使用的穷举搜索的策略，这很大程度的提高了目标检测的性能。但是其还存在一些缺陷，采用 Selective Search 算法提取的候选区域框数量多达 2000 个，并且要对每个框都要提交到 CNN 卷积神经网络进行提取特征和 SVM 分类操作，这带来了很大的计算量，大大降低了 R-CNN 的检测速度，也占据了大量的磁盘空间。另外，为了适应全连接层的输入要求，对候选区域需要进行归一化处理，这可能会导致该候选区域内的信息丢失。

一个改进的思路是，可以对整个图像做一次 CNN 特征提取，然后将候选区域框在原图的位置对应到 CNN 特征图上，然后将每个框的 CNN 特征输入到全连接层做后续计算。做了上述改进之后，对于一张图片只需要提取一次 CNN 特征。然而还存在的一个问题：每个候选区域框的大小尺寸是不一样的，而 CNN 全连接层输入长度的只能是固定的。SPP Net 的提出正好解决了上述问题。

2. SPP-net[15]

SPP-net 目标检测就是以 R-CNN 为基础并对其做出改进，主要有以下两点：一是取消了归一化处理，所以也不会出现信息丢失的问题，并且省去对归一化之后的图片的存储；二是在最后一层卷积层后新增了空间金字塔池化层，金字塔池化的意义就是将输入的任意尺寸大小的特征图都经过计算转换成固定大小的特征向量，然后将转换后的特征向量输入至全连接层。

在此之前的 CNN 中，输入图像的尺寸以及输出的结果一般都是固定大小和长度的，引入空间金字塔池化层的 SPP-net 使得网络可以接收任意尺寸的图像而

保持输出的长度不变。

SPP-net 进行目标检测的主要步骤如下：

(1)输入测试图片；

(2)选取候选区域：与 R-CNN 目标前两步相同，SPP-net 也是利用选择性搜索算法来选取图片的候选区域；

(3)特征提取：与 R-CNN 不同，之前 R-CNN 要对每个候选区域进行卷积计算得到卷积特征，SPP-net 对整张图片进行一次卷积运算，然后将 2000 多个候选区域对应的位置映射到卷积计算后的特征图上，通过这个改变，节省了大量的计算；

(4)分类与回归：这一步也与 R-CNN 相似，采用 SVM 对候选区域特征进行分类，然后通过微调目标边界框输出；

通过上述改进，SPP-net 解决了重复卷积计算的问题，在速度上有了很大改进，可以达到比 R-CNN 快一百倍。但是 SPP-net 仍然存在和 R-CNN 相同的缺陷：一是训练过程比较繁琐，要分成多个步骤，还要另外将特征存储在磁盘上；同时，SPP-net 的微调只是更新金字塔池化层后面的全连接层，而且在选取候选区域时仍然花费了很长的耗时。

3. Fast R-CNN

Fast R-CNN 是 R-CNN 的升级版，它的提出主要是解决 R-CNN 和 spp-net 不足之处，在保证检测效果的同时提高检测效率。Fast R-CNN 在 R-CNN 的基础上在最后一个卷积层后增加一个 ROI(Region of Interesting)池化层，ROI 指的是在对图片卷积后生成的特征图上的框,而 ROI 池化层就是针对 ROIs 设计的池化层，它其实是 SPP-net 空间金字塔池化层的简化版本，特点是输入图的尺寸可以是不同的，但是输出的尺寸是固定。同时，Fast R-CNN 采用非线性分类器 SoftMax 以多任务学习的方式使得分类和回归可以同时进行。与 R-CNN 相比，Fast R-CNN 不需要将训练和测试过程生成的中间值进行存储，因此在速度上取得了很大的提升。

Fast R-CNN 的进行目标检测的主要步骤如下：

(1) 输入测试图片；

(2) 特征提取：对整张图片进行卷积计算；

(3) 区域提名：使用选择性搜索算法获取候选区域，并且将选取的候选区域映射到特征图；

(4) 候选区域归一化：将每个候选区域对应的特征图输入到 RoI 池化层，并输出统一大小的特征表示；

(5) 分类与回归：与 R-CNN 不同，Fast R-CNN 放弃了 SVM 而采用 softmax 网络进行最后的分类，然后再进行回归微调输出最后结果。

4. Faster R-CNN

Faster R-CNN 对 R-CNN 进行了更进一步的改进，在 Fast R-CNN 的基础上，将原来的选择性搜索选取候选区域的方式，改成用 RPN(Region Proposal Network)区域生成网络获取候选区域，从而大大减少了时间开销。RPN 中采用 anchors 机制。Anchors 是尺寸固定的边界框，将不同尺寸和比例的 anchors 放置图片中得到大小不同的参考框，训练和检测过程，使用相对于参考框的偏移量来确定目标的位置。Faster R-CNN 可以看做是由 RPN 区域生成网络提取候选框模块和 Fast R-CNN 的检测模块组成的。同时，Faster R-CNN 中区域提名、分类以及回归等均共同使用 CNN 特征，从而进一步改善了检测速度。

Faster R-CNN 进行目标检测的步骤如下：

(1) 输入测试图片；

(2) 特征提取：与 Fast R-CNN 相同，将整张图片作为输入到卷积神经网络中，得到特征层；

(3) 选取候选区域：采用 RPN 区域生成网络获取候选区域。RPN 网络对输入图片的大小没有限制，其核心思想是直接采用卷积神经网络生成 Region Proposal；

(4) 分类与回归：利用 softmax 分类器进行计算分类，再进行回归得到最后的精确检测结果。

### 2.3.3 One-stage 目标检测算法

One-stage 算法没有 region proposal 阶段，直接得出检测到的目标结果概率及位置坐标信息，代表的算法有 YOLO 系列和 SSD。

1. YOLO 系列

YOLO 算法基于一个单独的 CNN 模型实现了端到端的目标检测，在基于 R-CNN 系列目标检测算法中，CNN 主要用来对图片中的目标进行分类，并没有采用 CNN 网络实现目标定位的功能，而 YOLO 仅使用 CNN 网络同时实现了目标的识别和定位。YOLO 的训练和检测阶段均在一个单独的网络上进行，不需要 region proposal，其将目标检测问题转换成回归问题来进行求解，使得检测速度有很大的提升，每秒可处理 45 张图片。

YOLO 检测网络由 24 个卷积层和 2 个全连接层组成。图像特征的提取由卷积层完成，而全连接层负责预测计算物体位置和分类概率。

YOLO 进行目标检测的简要步骤如下：

(1) 输入图片并调整图片大小，统一图片大小；

(2) 运行 CNN，将整张图片输入网络；

(3) 利用非极大抑制算法，得到最后的检测结果。

YOLO 将统一大小的图片划分成 S*S 个方格，每个方格负责检测各自范围内的物体，如果某个方格内位于或包含某个物体的中心位置，那么这个方格就负责对该物体的检测。每个方格会输出包含坐标、高度、宽度和置信度的边界盒信息，然后将该置信度与预定好的阈值比较，判断的是该边界盒是否包含某个物体及物体位置的准确度。

YOLO 的检测速度快、通用性强，而且整个训练和测试过程以整张图片为基础，因此对背景的误检率比较低。但与 R-CNN 系列物体检测算法相比，定位的准确性相对较差，且召回率低。

YOLOv2 则针对上述缺陷提出了优化方案，在保证检测速度的同时，提升了检测的准确度。YOLOv2 不是通过加深或加宽网络来追求更高的准确度，而是提

出了一系列的改进方案，反而使得网络较之前有所简化。YOLOv3 主要是借助一些其它算法中比较优秀的方案与 YOLO 相结合，使用更深层次的网络结构，进一步提高了检测精度，提升了对小目标的检测能力。

2. SSD

针对 YOLO 定位准确性低的问题，SSD 基于 YOLO 的回归思想采用 Faster R-CNN 的 anchor 机制，其获取目标位置以及物体类别的方法与 YOLO 一样都是采用回归，但 YOLO 预测位置时是基于整张图片而 SSD 对某个位置进行预测时基于的是该位置周围的特征。SSD 的核心设计理念主要有：1）检测时采用多尺度特征图。2）检测时利用卷积进行检测，YOLO 是最后一层检测结果由全连接层给出，而 SDD 直接采用卷积提取最后的检测结果。3）设置先验框。SSD 参照 Faster R-CNN 中的 anchor 机制，为每个方格设置尺度及长宽比不等的多个先验框，以这些先验框为基准预测物体边界框，一定程度上可以减少训练的难度。对每个先验框都输出一组检测值，即对应一个边界框，检测值包含各个类别的置信度以及边界框的位置坐标，置信度最高的类别就是该框所属的分类。

SSD 在保持了 YOLO 的高检测速度同时，大大提高窗口预测的精准度。

## 2.4 YOLOv2 目标检测算法

经过实验测试，本文选用 YOLOv2 网络进行 TX2 上的目标检测实现和研究，第三章将具体说明实验测试分析。

为了保持检测速度的优势同时优化 YOLO 网络的检测精度，YOLO 团队对原有的网络实行了一系列改进方法，提出了 YOLOv2 版本。

首先，YOLOv2 在每一个卷积层后面使用 Batch Normalization（批归一化）[24]，规范网络中每层的输入分布，有效的改善了收敛速度且减少了对其它正则化方法的使用，在原有 YOLO 的基础上提升了 2%的 mAP。之后，通过微调网络提升输入的分辨率，取得了 4%的 mAP 的提升。第三轮改进措施是采纳 Faster R-CNN 的 Anchor 机制，使用滑动窗口在卷积特征图上进行采样。原有网络在使用全连接层预测边界框时的 reshape 操作导致空间信息的丢失，使得定位准确度

较低。本轮改进去掉了全连接层和最后的池化层，缩减网络调整图片输入分辨率为416×416，引进 anchor boxes 同时预测目标的分类和定位。结果使得模型的召回率从 81%提升到 88%，而 mAP 只有小幅度的下降。随后，通过 Dimension Clusters（维度聚类）和 Direct location prediction（直接位置预测）对 anchor boxes 进行改进，使 mAP 提升了 5%。为了提高对小目标的检测效果，通过增加一个 passthrouph layer（转移层），将浅层特征图与深层特征图连结，使最后检测的特征图拥有更好的细粒度特征，从而取得了 1%的性能提升。另外，通过 multi-scale training（多尺度训练），提高网络对图片尺寸的鲁棒性。

为了使网络速度更快，YOLOv2 采用 darknet 框架[32]实现，提出了 Darknet-19 分类模型，Darknet-19 有 19 个卷积层以及 5 个最大池化层。darknet 是开源的深度学习框架，完全基于 C 语言和 CUDA 实现，同时支持 CPU 和 GPU。另外，darknet 没有任何依赖项，灵活性高，比较适合对网络底层的研究，方便从底层对网络进行改进。

最终形成 YOLOv2 整体网络结构如图 2-5 所示，由 darknet-19 网络去掉最后一层卷积层以及后面添加的检测网络组成，共有 32 层。

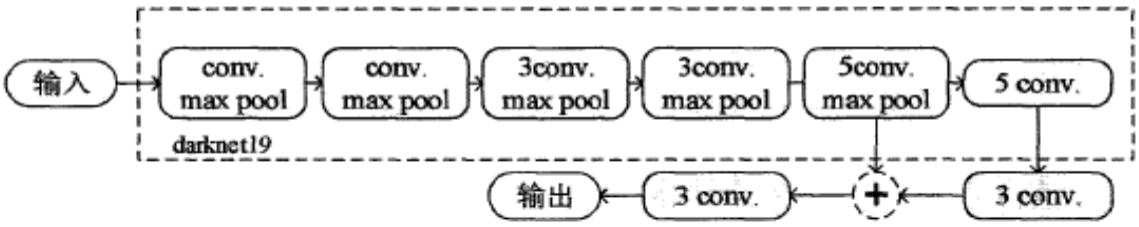


图 2-5 YOLOv2 整体结构图

## 2.5 本章小结

TX2 的高性能计算主要得益于其先进的 GPU 结构，本章简要介绍了 TX2 平台，并对 CPU 与 GPU 的异构进行了分析。另外，在调查资料的基础上，对目标检测技术的发展情况进行概述。传统目标检测技术采用滑动窗口遍历整张图片的穷举策略造成了非常高的复杂性，而且其采用手工设计的特征鲁棒性比较低。region proposal 区域提名方法的提出，很好的解决了采用滑动窗口所出现的问题，

而卷积神经网络的引进开启了目标检测技术的新篇章。本章的最后对本文采用的YOLOv2 网络进行了详细的分析。

# 第 3 章 目标检测模型选择与分析

当前，目标检测模型发展迅速而且应用广泛，很多目标检测模型在应用中都表现了很好的性能，为选出适合本文实验平台 TX2 的模型，将性能突出的几个主流模型部署到实际平台上进行实验测试，并比较分析不同网络模型的准确度和速度性能指标。

## 3.1 性能指标

首先介绍一下几个主要指标，从而便于对目标检测算法的衡量和评估。

为了评估目标检测的准确度，提出了一个概念 IoU(Intersection over Union)，它表示的是预测框与真实框之间的重叠程度，它的值介于 0 到 1 之间。

模型准确度的总体评估可以用整体的平均 IoU 表示，计算方式如公式 3-1。

$$R_{IoU} = \frac{\sum_{i=1}^{I} IoU_i}{I} \tag{3-1}$$

其中 I 是用于测试的图片的总数，即测试集中所有图片 IoU 的平均值。

此外，在对每一次目标检测进行评估时，一般设置一个 IoU 阈值，当某次预测的 IoU 值大于设定的阈值时，该预测框才被认定为 TP（真阳性），反之就是 FP（假阳性）。而准确率可以由 precision=TP/（TP+FP）计算得到。

还有一个主要指标则是衡量目标检测模型的速度，用 FPS（帧率）表示，它指的是每秒钟所检测的图片数。

另外经常采用 mAP(均值平均精度)作为目标检测模型的评价指标，它的值越大表示该模型的准确度越高。

## 3.2 目标检测模型对比

论文[17]中对各目标检测技术的性能进行了比较分析，训练集采用 VOC2007

和 VOC2012，测试集使用 VOC2007，不同模型对比得到表 3-1 对比结果。

表 3-1 各目标检测技术基于 VOC2007 数据集的对比

| 检测模型 | mAP | FPS | 检测模型 | mAP | FPS |
|---|---|---|---|---|---|
| Fast R-CNN | 70.0 | 0.5 | YOLOv2 288*288 | 69.0 | 91 |
| Fast R-CNN VGG-16 | 73.2 | 7 | YOLOv2 352*352 | 73.7 | 81 |
| Fast R-CNN ResNet | 76.4 | 5 | YOLOv2 416*416 | 76.8 | 67 |
| YOLO | 63.4 | 45 | YOLOv2 480*480 | 77.8 | 59 |
| SSD300 | 74.3 | 46 | YOLOv2 544*544 | 78.6 | 40 |
| SSD500 | 76.8 | 19 | Tiny-YOLO | 57.1 | 207 |
| DPM | 34.3 | 0.5 | | | |

另外，还采用相同的环境和数据集分别对 tiny-YOLO 以及传统目标检测技术的代表 DPM 算法进行了测试。DPM 的测试结果对应的 mAP 为 34.3，检测速度为 0.5fps。由此可以看出，与传统的目标检测技术相比，基于深度学习的目标检测技术在准确度上要高的多，而且在以 YOLO 为代表的端到端的检测技术提出之后，目标检测的速度有了质的飞跃。显而易见，基于深度学习的目标检测技术无论是准确度还是速度都远远超过了传统的目标检测技术，基于深度学习的目标检测技术已经成为主流。此外，由表 3-1 可以看出，YOLOv2 的在速度上较之前的目标检测技术有了很大的提升，为了使结果更加具有说服力，采用了不同大小的输入图片进行测试。此外，tiny-YOLO 是 YOLOv2 的轻量级版本，它的网络结构只有 16 层，所以虽然检测精度差强人意，但是其速度几乎是完整 YOLOv2 网络的 3 倍。

本文实时目标检测系统选择基于嵌入式平台 TX2 实现，因此还需在 TX2 上进行实验测试选取最适合的目标检测模型。不同的平台在网络计算过程中的计算量时一样的，因此嵌入式平台上模型的准确度几乎没有影响，主要时由于有限的资源导致的计算速度的下降，使得检测速度下降。根据前文对不同目标检测技术的准确度和速度分析，在 TX2 平台上采用相同的数据集对 Faster R-CNN、YOLOv1、YOLOv2、SSD 以及 tiny-YOLO 进行进一步实验。测试结果如图 3-1 所示。

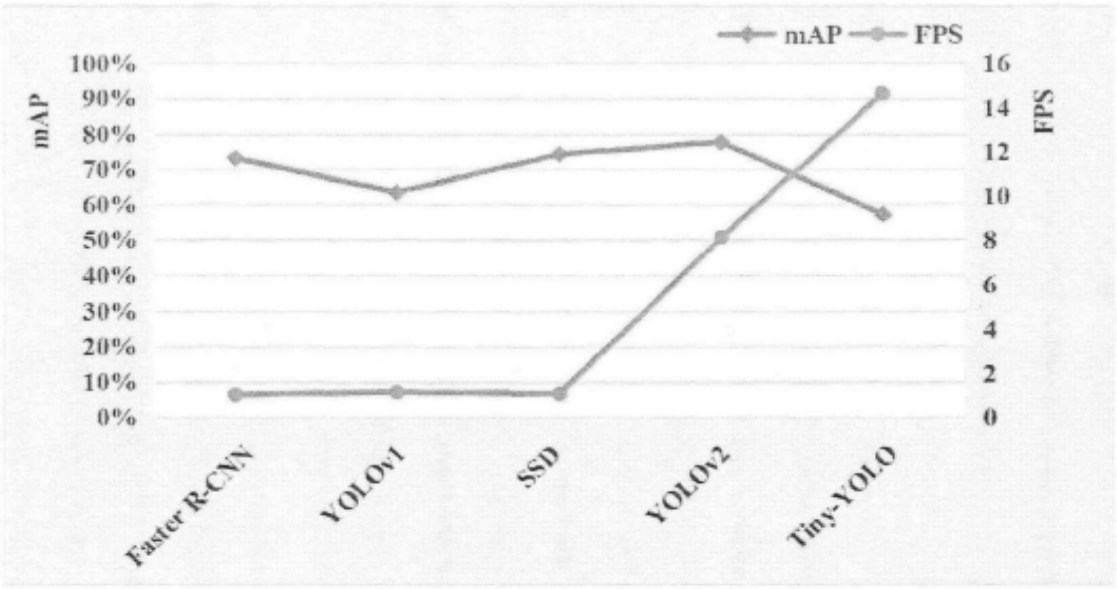


图 3-1 TX2 上模型对比

由实验结果可以看出，SSD 在 TX2 平台上的检测速度非常低，这是因为 SSD 模型结构比较复杂，所以在 TX2 平台上实现时，有限的计算资源使其检测速度受到很大限制，不能达到实时检测的速度要求。考虑到还要保证系统可靠性，检测的准确度不能太低，经过权衡准确度和速度两个指标，最终确定采用 YOLOv2 模型。在此基础上设计方案对其进一步优化。

## 3.3 基础网络选择

YOLOv2 模型结构由两部分组成，包括用于提取特征的基础网络以及负责检测的检测网络。其中基础网络占据了整个目标检测的大部分计算时间，因此基础网络的选择对整体性能有着重要的影响。

考虑到本文的应用平台是资源有限的嵌入式设备，因此尝试使用轻量级的 CNN 网络架构进行特征提取来减少计算量，从而节省计算资源来提升检测速度。选择的轻量级 CNN 网络是现有比较先进的 mobilenet_v1、mobilenet_v2、shufflenet[41][42]，上述网络架构都是采用深度可分离卷积，应用到 YOLOv2 的特征提取后，测得如表 3-2 结果。

表 3-2 不同基础网络性能对比

| CNN | FPS | IoU | param size |
|---|---|---|---|
| darknet-19 | 12 | 0.69 | 50M |
| mobilenet_v1 | 16 | 0.61 | 24M |
| mobilenet_v2 | 25 | 0.57 | 12M |
| shufflenet | 13 | 0.65 | 47M |

由表看出，虽然轻量级的 CNN 网络架构在检测速度上有明显的优势，但准确度相对较低，因此在不同的目标检测系统设计中，根据各自追求的速度和准确度采用不同的基础网络。而本文的优化方案在实现时，要以一定的准确度代价来换取速度的提高。因此选用原始的基础网络为 darknet-19 的 YOLOv2 作为本文的研究模型。

## 3.4 YOLOv2 训练与测试

本文主要针对网络检测过程的进行优化研究，因此可以利用高性能的实验环境进行网络的训练。为提高网络训练的效率，选择在具有更高计算性能的计算机上搭建与 TX2 相同的实验环境，进行网络的训练，装有相同环境和网络的 TX2 可以直接利用训练得到的权重文件进行测试。在检测过程中，可以结合 python 接口的调用，实现模型对视频文件输入的检测。

1、软硬件平台搭建

本课题网络训练时使用装有 GTX 1080Ti GPU 的计算机，操作系统是 Ubuntu16.04，测试环境是装有 Ubuntu16.04 的 TX2 平台。

分别在两个平台上安装相同的相关依赖环境。各软件或驱动版本选择如下：NVIDIA 显卡驱动 NVIDIA-Linux-x86_64-375.66.run、cuda_8.0、opencv_3.1.0、cuDNN_v6.0、python2.7。完成上述软件的安装后，下载安装 darknet。

2、数据集准备

在目标检测的研究，经常采用 PASCAL VOC，ImageNet 以及 COCO 数据集

等。本文第四章采用的是 DAC 会议主办的低功耗目标检测系统设计挑战赛提供的数据集。而在实际系统的设计中，根据特定的系统需求，可定做有针对性的数据集。

图片获取：由于网络训练需要的数据集规模庞大，为了提高效率，采用 ffmpeg 工具，将拍好的视频转成图片 jpg 文件。

图片标记：图片标记使用 labelImg 工具。利用该工具可以在图形用户界面对图片进行手动标记，并生成相应的描述图片的 xml 文件。

格式处理：将标记的图片整理为 YOLOv2 可用的格式，把标记图片生成的 xml 文件放在 Annotations 文件夹中，把原图放在 JPEGImages 文件夹中，并且让每个 jpg 图片对应一个 xml 文件。Main 文件夹中存放 train.txt，test.txt 等文件，用来指定训练的数据和测试的数据，文件内容是不带后缀的图片名称列表，在这里只要训练用到的 train.txt 文件。最后执行 voc_label.py 生成 label 文件。根据代码准备的数据集的目录结构如下：

表 3-3 数据集目录结构

```
--VOC
    --Annotations
    --ImageSets
      --Main
    --JPEGImages
```

3、网络训练

为了加快提高网络的训练效率，选择在性能更高的计算机上进行网络的训练。在开始训练前，首先要准备好 yolov2-voc.cfg voc.names voc.data 三个配置文件，其中 voc.cfg 文件描述了网络结构以及相关参数的设定，voc.names 文件放在 /darknet/data 目录下，用于指定类别，voc.data 文件放在/darknet/cfg 目录下，用于指定类别数量和训练样本。

完成配置文件的修改即可开始训练，可以利用 YOLO 官网提供的预训练权

重进行训练，也可以不采用预训练权重直接训练。训练的过程中可以随时停止，在指定目录下会生成相应的权重文件，之后还可以继续训练，继续训练时，之前生成的权重文件就相当于预训练权重。

网络训练完成后，将权重文件复制到安装了相同环境的 TX2 平台上。

4、模型测试

训练好的 YOLOv2 模型，每检测一张图片都要手动执行一次检测命令，而实际应用中往往输入的是视频文件，要实现对连续的视频帧进行检测。因此，可以采用 python 接口来不断调用 YOLOv2 模型，实现流水线式的目标检测任务。

## 3.5 本章小结

基于前文对各种目标检测技术的学习，在了解相关的性能指标后，在 TX2 平台上，通过对几个主流目标检测模型的实际测试，对比各模型的准确度和速度指标，选取最适合 TX2 平台的模型。通过实验对比，最终采用基础网络为 darknet19 的 YOLOv2 目标检测模型。在选定开发平台和模型的基础上，完成了实验环境的搭建，详细介绍了网络模型训练的前期准备和训练过程，并对训练好的模型进行测试。通过测试发现，现有模型已经具备了良好的准确度，满足实际应用的高精度需求，但是其在 TX2 平台上的检测速度只有 10fps 左右，远远达不到实时检测的设计初衷。因此还需对其进行进一步的优化，来提升检测速度，同时还要保持准确度的优势。

# 第 4 章 目标检测网络优化方法研究

本章主要根据网络结构以及平台的体系结构特点进行网络优化，优化的目标即进一步提升网络的检测速度，同时还要保持较高的准确度。

## 4.1 数据集

本章实验采用的数据集来源于 2018 年 DAC 主办的低功耗目标检测系统设计挑战赛。该数据集的图片由大疆无人机提供的，共计 93520 张图片，包含船、火车等 12 个大类，公分为 95 个子类。该数据集的特点是每张图片中只有一个目标物体。

## 4.2 网络参数及结构调整

要进行网络模型的优化，最简单的就是通过参数的调整来使网络达到最适合的状态。然后再进行网络的压缩来进一步加速网络。

### 4.2.1 调整参数

在训练过程中要不断的调整相关参数来达到当前数据集最好的网络性能。

1.输入尺寸。不同输入大小对网络的训练和检测过程都有影响，我们通过实验来探索图片输入大小对 TX2 上的 YOLOv2 的影响，结果如下图 4-1 所示。检测过程中，图片越大其准确度越高，但速度会下降。经过对两个指标的权衡，最终采用 416×416 的分辨率。

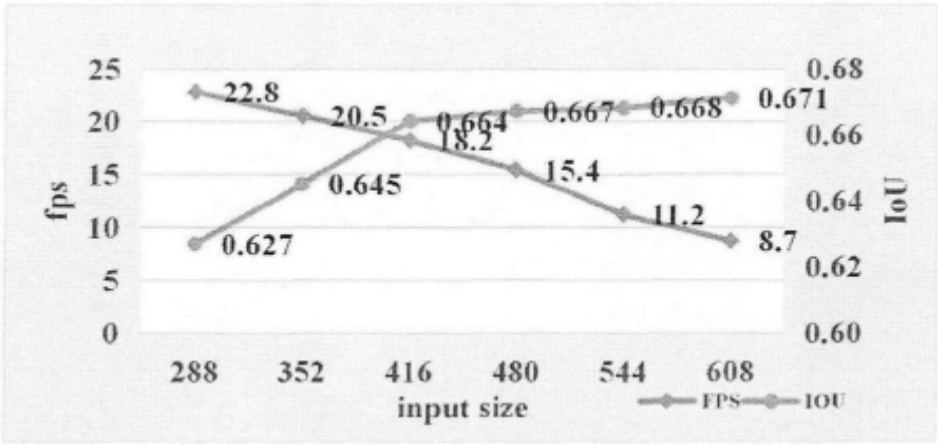


图 4-1 图片大小对模型的影响

2.学习率。学习率对网络模型的学习过程有调节的作用，对训练结果有很重要的影响。学习率如果太小，就会导致模型的收敛速度很慢，从而带来大量的运算，如果太大，可能会使损失在极值点两侧震荡，导致损失无法梯度下降。因此，在迭代的过程中观察损失的变化情况，来不断的调整固定学习率。经过多次尝试和探索，将学习率的初始值设为 0.001，为了增大初始训练时损失的下降速度，在 100 次迭代完成后，将学习率扩大 10 倍继续进行迭代，当次数到达 80000、1200000、160000、220000 时，将学习率分别乘 0.1 来防止学习率过大。在 cfg 配置文件的相关值设置如表 4-1 所示。通过学习率的调整，检测网络的准确率有了明显的提升。

表 4-1 学习率参数设置

```
learning_rate=0.001
burn_in=1000
max_batches =300000
policy=steps
steps=-1,100,80000,120000,160000，220000
scales=0.1,10,0.1,0.1,0.1,0.1
```

### 4.2.2 网络结构调整

为了进一步提升网络速度，减少网络运算量，将 YOLOv2 的结构进行进一步的探索和调整。经前文对 YOLOv2 结构的分析，其详细的网络结构如图 4-2

所示。

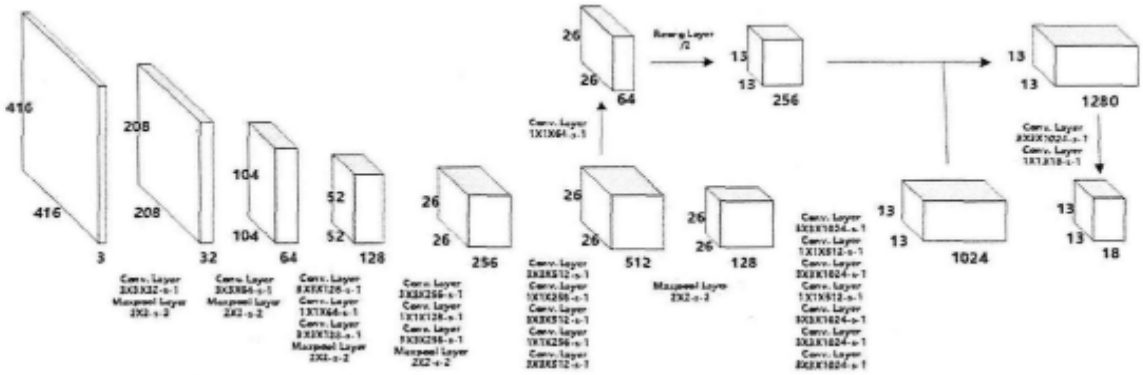


图 4-2 YOLOv2 网络结构细节图

在对网络结构的调整中，核心思想是通过减少网络层数和卷积核 filter 个数，来减少计算量从而达到更快的检测速度。YOLOv2 使用了多个大小 3×3 的卷积核进行卷积提取特征，紧随其后的是用来整合不同维度特征的大小为 1×1 的卷积核。结合最后输出尺寸的要求，通过减少所有卷积层的 filter 个数以及部分计算量较大的卷积层，来减少计算量，在减少运算度的同时尽可能的保留网络结构。

经过多次调整和测试的尝试，测得如图 4-3 结果，由于网络结构的复杂性，只标明了网络层数。

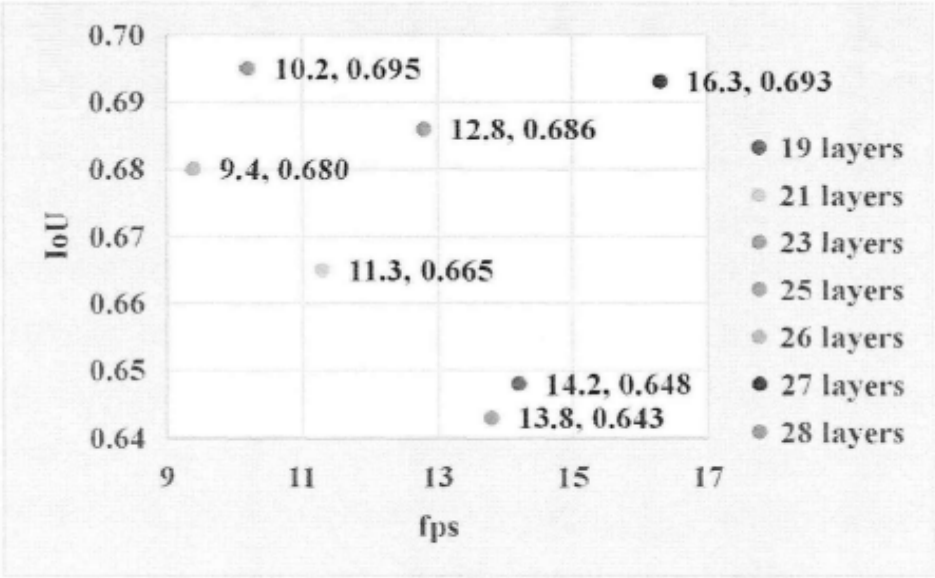


图 4-3 不同网络结构的性能

根据上图 4-3 结果，调整后的 27 层网络无论在准确度和速度上都具有一定

的优势，因此最终选用了如下图所示的 27 层网络结构，为了在目标检测时获得更高的 IoU，将输出层的尺寸增加到 26×26×18。

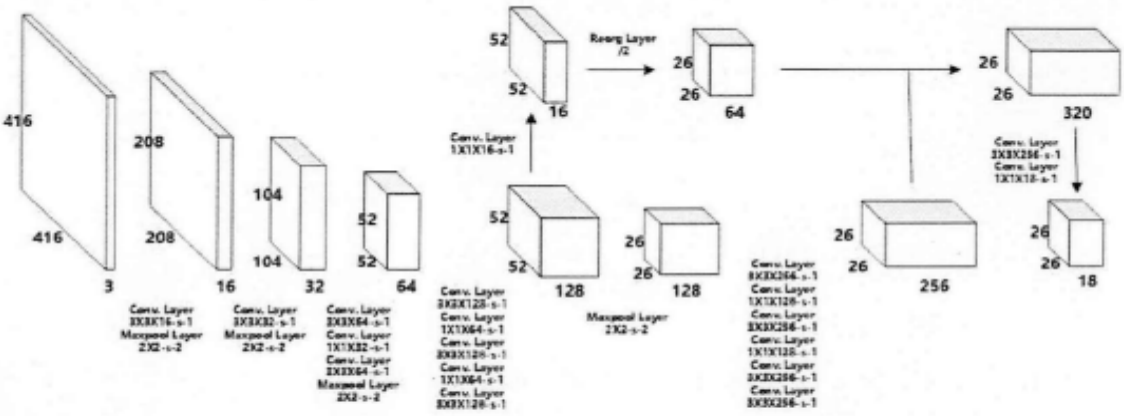


图 4-4 调整后的 YOLOv2 网络结构

## 4.3 半精度浮点数的应用

半精度浮点数使用 16 位存储，具有较少的位数，所以能够表示数据的精度和数据的范围都相对较小，因此相应的参与计算的位数以及计算所需的时间相应就会少，同时也减小了存储的压力。所以与单精度浮点数相比，半精度浮点数的使用可以节省一半的带宽和存储空间，但是要牺牲一定的精度。半精度浮点类型的提出在实际应用中的作用非常大，它使得应用程序可以处理的数据集更大，同时也可以减少存储和计算压力从而获得更高的性能。因此半精度浮点数的使用对一些大规模的神经网络以及一些受存储带宽限制的信号处理内核非常有意义。

另外，半精度浮点数在深度学习中的应用已经比较广泛，NVIDIA 已经提供了相应的软硬件的支持。从 CUDA7.5 开始支持对半精度浮点数的存储和计算，并添加了 half 和 half2 两种数据类型，且新增了两个转换函数__half2float()和__float2half()用于 FP16 和 FP32 之间的转换，同时还增加了混合精度的矩阵乘法运算等。混合精度在 Google 的 TPU 上也有应用，采用了 8 位低精度计算，极大的提高了计算效率，而结果表明低精度运算带来的算法准确率损失很小。所以针对一些对精度要求不高的计算，可以采用半精度甚至低精度运算，从而显著提升推理速度降低推理过程的计算难度。

因此本节优化方案采用混合精度的计算来对系统进行改进，在 GPU 模式下运行本文实现的检测模型时，首先需要把经过预处理的图片信息和参数等拷贝到 GPU 上，然后由 GPU 对其进行运算。而整个过程所涉及的数据大多是 32 位浮点类型，所以会花费大量的时间进行拷贝，并且 GPU 的运算量也很高。

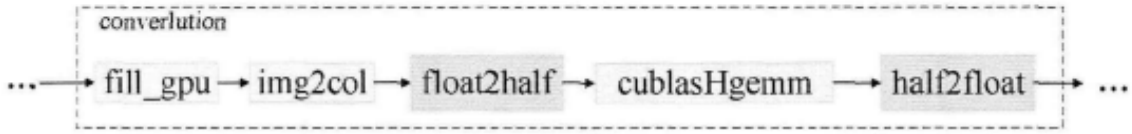


图 4-5 卷积层的半精度转换

卷积层将卷积运算转换为矩阵乘操作，这些矩阵乘是整个模型中计算量最大的部分，所以首先想到将将 32 位浮点数的矩阵乘操作转换为 16 位的矩阵乘，在对图片的检测过程，卷积层运算只有正向传播过程，所以在卷积层的 kernel 函数中，添加了使用半精度浮点数的正向传播函数，并在卷积层的矩阵乘与其他函数之间提供 32 位浮点数和 16 位浮点数的转换操作。实验统计了矩阵乘、数据转换的时间，测试结果如图 4-6 所示。

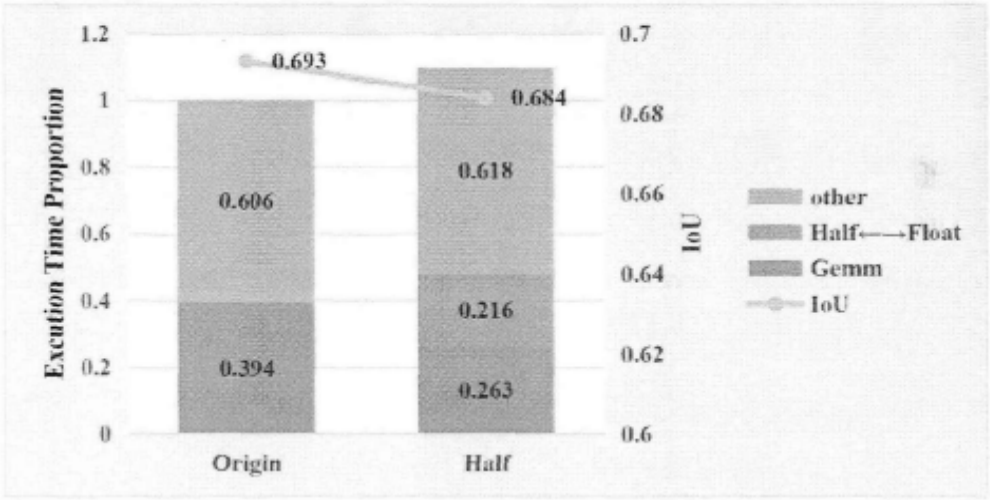


图 4-6 卷积层采用 half 前后结果

结果表明经过改进后，由于数据位数的减少，模型的准确度稍有下降，但下降幅度很小在可接受的范围之内。整个过程网络中矩阵乘运算所占用的时间确实有了明显的下降，但由于网络中卷积层的数量比较多，每次卷积运算就需要进行数据转换操作，频繁的数据转化导致了时间的浪费。

因此，对该方案进行调整，在预处理后的图片数据进入网络之前，先进行数

据的转换，将浮点数转为 half 类型，然后再将转换后的数据拷贝到 GPU 中参与网络中的计算，这样在网络的计算过程中，不再需要数据类型的转换，只要在网络输出结果之前，再将数据转换回 32 位浮点数。该方案不仅降低了网络中的计算量，节省计算时间和资源，而且使得在拷贝的过程也节省了很多的数据量。

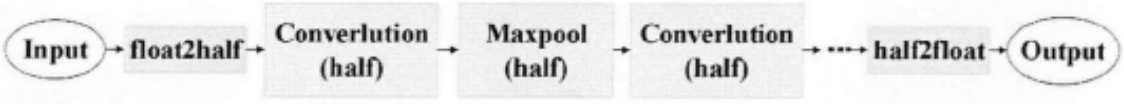


图 4-7 整体网络半精度转换

根据调整后的方案，将网络中的所有层的单精度的浮点数运算转换成半精度的浮点数运算，使用 half 前后的模型对比如图 4-8 所示，速度较改进之前有所提升。

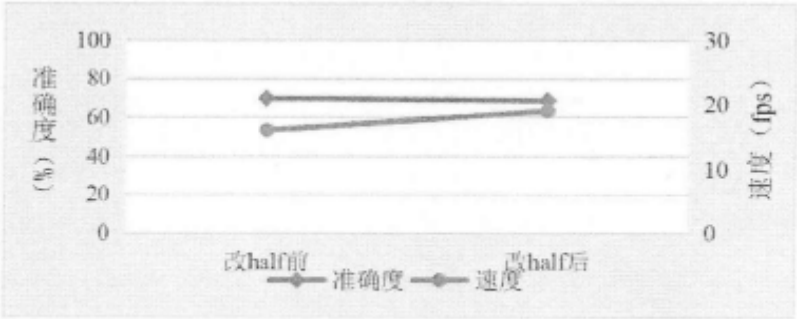


图 4-8 使用 half 前后对比

## 4.4 多线程 CPU/GPU 流水设计

在计算系统对目标物体的检测时间时，从读取图片开始计算，到系统检测完成并返回目标及其位置结果结束。这个时间段内，代码在 CPU 和 GPU 上的执行情况如图 4-9 所示，模型加载及读取图片并进行预处理在 CPU 上执行，然后 GPU 进行网络中的运算和预测，最后由 CPU 返回检测结果。利用时间检测函数来统计代码各部分的运行时间，发现目标检测过程的时间主要花费在 CPU 图片预处理和 GPU 的网络预测阶段，而最终 CPU 输出的检测结果的时间非常小，可以忽略不计。

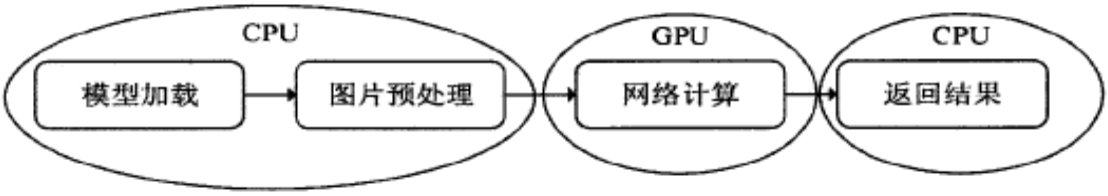


图 4-9 检测过程 CPU 和 GPU 的任务分配

通过进一步统计分析 YOLOv2 网络在 CPU 和 GPU 上执行的情况，如图 4-10 所示，在单线程运行时，处理每一帧的时间大约是 50ms，也就是相当于每秒 20 帧的速度，其中在 GPU 上执行的时间是 30ms，在 CPU 上执行的时间是 20ms。

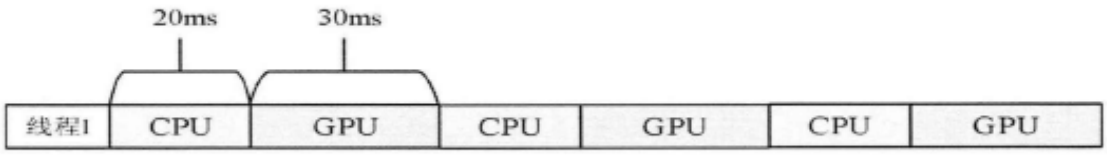


图 4-10 单线程 CPU 和 GPU 的运行情况

考虑到在 TX2 开发板上是多核的 CPU，所以尝试通过开启多线程的方式来进行调度，尽量让 GPU 处于一直计算中，从而最大化的利用 GPU 的计算能力。

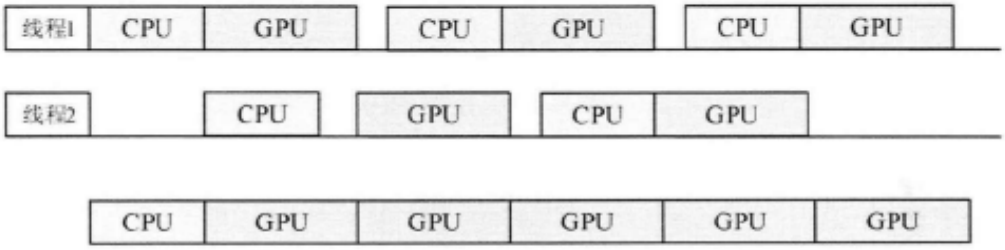


图 4-11 多线程 CPU 和 GPU 的运行情况

因此设计了图 4-11 所示的调度方式，让一个线程在执行 GPU 任务的同时，另一个线程先开始 CPU 的图片读取和预处理任务，等到第一个线程完成 GPU 的计算后，第二个线程就可以立即开始 GPU 的计算任务，整个过程将上一张图片的 GPU 计算任务和下一张图片的 CPU 预处理阶段同时进行，这样在检测时可以节省掉每一张图片的预处理的时间。根据数据集和输入要求的不同，开启的线程数量可以有所调整。

我们根据当前应用，使用两个线程进行流水线式的检测。所以最后整个过程花费的时间完全隐藏了 CPU 的处理时间，图片的检测时间只需要计算 GPU 的处理时间即可。也就是说检测每一张图片时间只需要花费一个 GPU 的处理时间

30ms，这样计算可以求出使用这种调用方式理论上速度可以达到 33fps，但是由于在测试 CPU 和 GPU 的处理时间时算的是平均数，所以多线程的调用方式与预期的理想结果有差距，但还是节省了很大一部分 CPU 的预处理时间。

实验测试结果如图 4-12 所示，本节方案改进过程不涉及到网络结构的变化，因此对系统的准确度没有影响，同时使检测速度提升了大约 4fps。

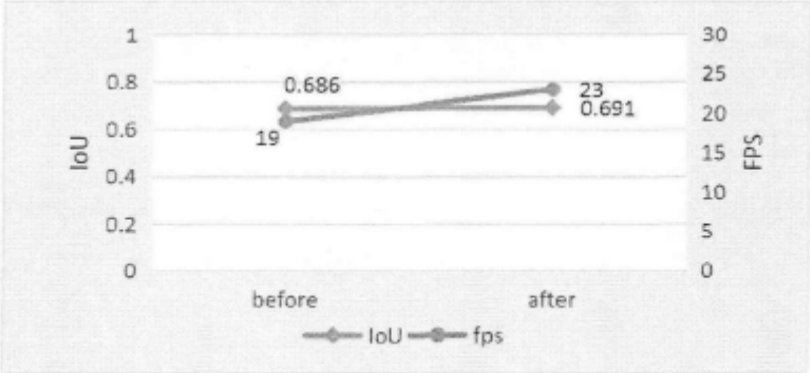


图 4-12 使用多线程前后对比

## 4.5 本章小结

原始的 YOLOv2 模型部署到 TX2 后，仍然具有较高的检测准确度，但在速度上远远达不到实时性的需求，因此本章通过设计一系列的优化方案，在保持准确度优势的前提下，对系统实现速度的提升从而实现对目标的实时检测。另外，本章优化方案主要针对 YOLOv2 本身的结构和实际平台的特点设计实现的，针对不同的数据集或者不同的基于 TX2 的目标检测应用场景具有通用性。

# 第 5 章 基于运动目标的检测优化研究

目标检测在实际生活的应用中，很多场景要求以视频作为输入，并对视频中的运动物体进行实时检测和追踪。因此可以采用目标追踪技术达到上述效果，且一些目标追踪技术的速度比目标检测的速度快的多。而受光照及遮挡等因素，目标追踪技术的准确率往往难遂人意。因此我们想到可以将我们的目标检测模型与目标追踪技术相结合，采用协同调度的方案，实现速度和准确度上严格的实际应用标准。

在实验探索时，我们首先想到的是 KCF（相关滤波跟踪算法），但是将 KCF 部署到 TX2 开发板上后，测试速度非常慢，只有大约 0.5fps，经过进一步分析，发现开源的 KCF 代码仅支持 CPU，在 TX2 上执行时无法利用计算性能较高的 GPU。因此，最后决定采用与 GOTURN[22]算法相结合。而对平滑运动的目标进行检测时，根据其运动平滑性的特点设计位置预测算法。

## 5.1 数据集

本章实验采用的数据集来自于对高空飞行中无人机的跟拍视频，采用 ffmpeg 工具将视频文件导出为图片后并进行手动标记。训练集由 3500 张图片组成。数据集图片中目标物体只有一个类别为无人机，其背景是不固定的。

## 5.2 与 GOTURN 目标追踪算法的协同调度

GOTURN 模型的基本原理是：基于前一帧目标的位置，确定当前帧的搜索区域（search region），经过回归得到最终位置结果。因此根据 GOTURN 网络的特点，提出与 GOTURN 的结合调度的方案：同时运行两个网络，利用 YOLOv2 确定目标的初始位置，并将该结果作为 GOTURN 的输入，然后迭代进行若干次 GOTURN 模型进行目标的跟踪。检测过程两种网络结合调用示意图如图 5-1。

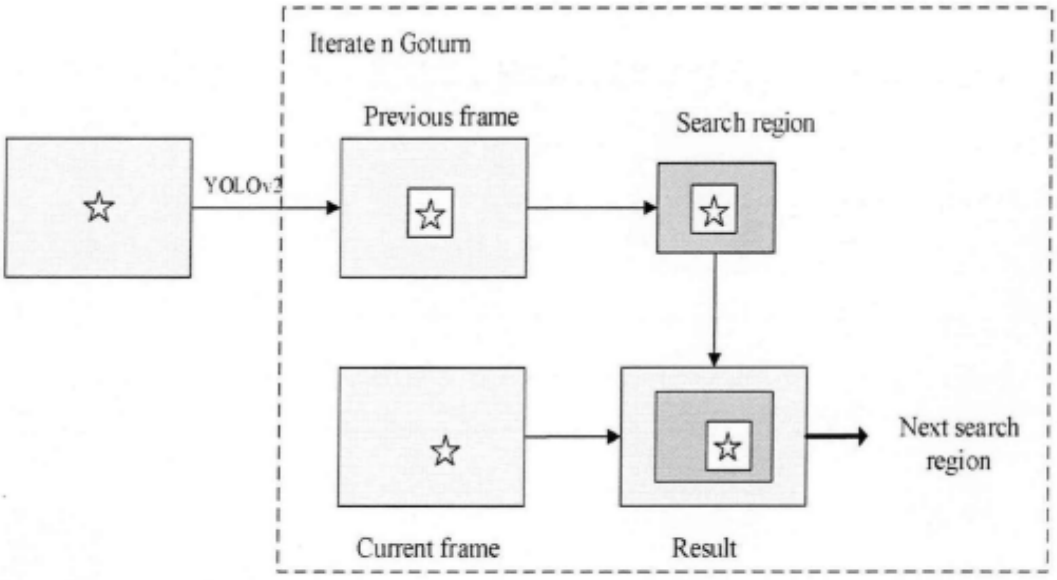


图 5-1 YOLOv2 与 GOTURN 调用示意图

首先通过 YOLOv2 网络来识别第一张图片获得目标位置，然后将第一张的结果作为 GOTURN 网络的输入，在 GOTURN 模型中迭代识别 n 张图片进行跟踪，跟踪 n 帧后，下一帧再回到 YOLOv2 网络中进行检测，来纠正跟踪产生的偏移从而确保检测的准确度。通过实验来确定 n 的取值，即确定每通过 YOLOv2 检测一帧后，使用 GOTURN 网络迭代跟踪几帧可以使得整体效果达到最好。YOLOv2 与 GOTURN 协同调度代码实现如表 5-1，通过进行多次实验，将 n 取不同的值来测试的取值对速度提升的影响程度。

表 5-1 YOLOv2 与 GOTURN 协同调度设计

```
for i in range(1):    // yolov2 检测 1 帧
    image[0, :, :] = image_draw[:, :]
    result = procfunc.detect(image, 1)
    location = result[0]
    image_last = image
for i in range(n):    // goturn 追踪 n 帧
    image[0, :, :] = image_draw[:, :]
    location = tracking.tracking(image_last, location, image)
    print(location)
    image_last = image
```

由于 GOTURN 网络速度快，因此 GOTURN 迭代的次数越多，系统的速度越快。首先通过实验测试了不同的检测和跟踪分配情况下的整体速度，表 5-2 给出实验对比结果，从表格中可以看出，对当前数据集，只采用 YOLOv2 网络时速度是 20.97fps，而只有 GOTURN 网络时的速度是 28.79fps。

表 5-2 YOLOv2 与 GOTURN 不同调用速度

| YOLOv2 | GOTURN | FPS |
|---|---|---|
| 1 | 0 | 20.97 |
| 1 | 1 | 22.34 |
| 1 | 2 | 24.05 |
| 1 | 5 | 25.24 |
| 1 | 8 | 25.91 |
| 0 | 1 | 28.79 |

为了保证系统的可靠性，进一步实验测试对比不同分配情况的准确度。在进行实验时，在训练之前，从无人机拍摄的视频中随机选取三个作为后续的测试集，并将视频导出为图片后进行标记，然后通过用系统识别生成的坐标位置结果与人工标记的结果进行对比，计算两者在不同的 IoU 阈值上的正确率，通过计算得出表结果。首先测试了 YOLOv2 网络基于上述测试集的准确率，为后续实验提供参照。

表 5-3 YOLOv2 网络的准确率

| IoU 阈值 | video1 | video2 | video3 |
|---|---|---|---|
| 0.3 | 1.0 | 1.0 | 1.0 |
| 0.5 | 1.0 | 1.0 | 1.0 |
| 0.7 | 0.938 | 0.627 | 0.60 |
| 0.8 | 0.592 | 0.236 | 0.118 |

从表 5-3 可以看出，当系统只用 YOLOv2 网络时准确率非常高，一般 IoU 阈值设置为 50%时，即预测框与真实框的重叠部分超过两框并集面积的一半时，正确率可以达到 100%。

经过测试，将 YOLOv2 和 GOTURN 的调用方式设置成 YOLOv2 每检测 1 帧图像，GOTURN 迭代跟踪 5 帧图像时，准确率统计结果如表 5-4 所示，对比

发现这种分配方式的准确率基本上与单用 YOLOv2 网络的结果差距不大，但是系统速度得到进一步提升。

表 5-4 优化后的准确率

| IoU | video1 | video2 | video3 |
|---|---|---|---|
| 0.3 | 1.0 | 1.0 | 1.0 |
| 0.5 | 1.0 | 1.0 | 1.0 |
| 0.7 | 0.9 | 0.754 | 0.445 |
| 0.8 | 0.49 | 0.245 | 0.064 |

## 5.3 基于平滑运动物体的位置预测算法

生活中，一些物体的运动往往具有一定的平滑性和规律性。在时间连续的前提下，其位置往往可以通过轨迹分析和数学计算进行预测，于是根据这种特性设计了位置预测算法，将其与 YOLOv2 网络相结合。

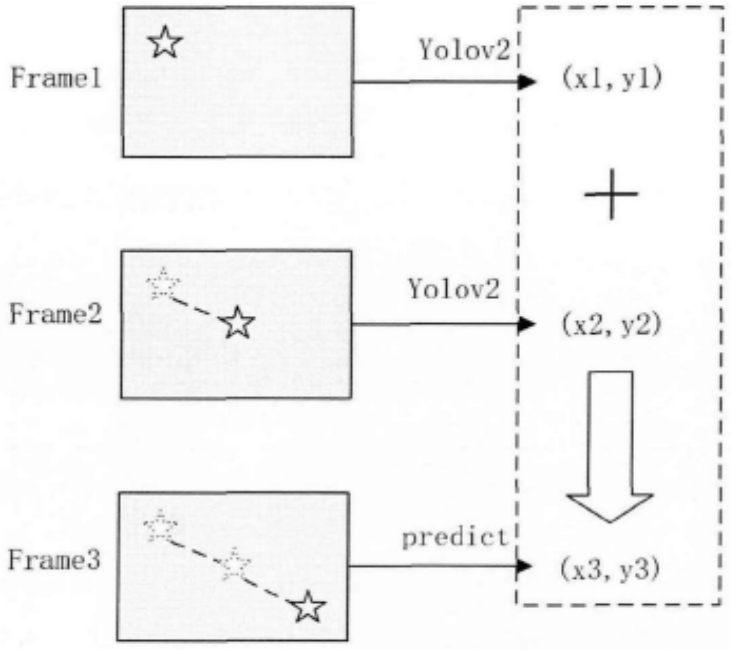


图 5-2 位置预测算法原理图

该算法的原理图如图所示，可以根据前两帧的识别结果，得到目标在这两帧的中心点位置，然后根据中心点的位置变化推测出目标在这一小段时间内的运动方向和运动速度，然后通过计算得出下一帧目标的预测位置。该算法的前提是，

假设平滑运动物体在连续三帧时间内为匀速直线运动。实现了该算法后，采用YOLOv2 网络识别前两帧然后调用位置预测算法识别后一帧的调用方式。实验表明，采用这种设计方案，系统的速度可以达到 30.4fps，进一步统计该方案的准确度，结果如下：

表 5-4 YOLOv2 和位置预测算法结合的准确率

| IoU 阈值 | video1 | video2 | video3 |
|---|---|---|---|
| 0.3 | 1 | 1 | 1 |
| 0.5 | 1 | 0.973 | 1 |
| 0.7 | 0.831 | 0.545 | 0.536 |
| 0.8 | 0.477 | 0.155 | 0.109 |

实验结果表明，当 IoU 阈值设为 50%时，三个数据集对应的准确率都能达到 95%以上。因此采用位置预测算法使得系统的速度提升显著，可以达到实时性检测的需求，而且准确率损失在可接受范围之内，达到了系统实际应用的标准。

## 5.4 本章小结

本章根据实际应用环境中运动目标的实时检测进一步提出了两种优化方案。在第四章优化方案的基础上，将改进的 YOLOv2 分别与 GOTURN 目标追踪算法和位置预测算法相结合，虽然最后的方案造成 IoU 微小损失，但仍然保持了较高的准确率。最后结合针对具有运动平滑性和时间连续性的运动物体设计的位置预测算法使得检测速度最高达到 30fps。上述两种方法主要适用于单个运动目标的检测，在推广运用上具有一定的局限性，但是取得了明显的速度提升，在对单个目标的实时检测和追踪问题上具有很大的意义。

# 第 6 章 总结与展望

随着嵌入式异构 GPU 平台的不断发展，以及多种应用对实时目标检测系统的需求越来越严格，使得基于嵌入式异构 GPU 平台的实时目标检测系统的设计研究具有非常高的理论意义和实际应用价值。

本文通过对各种目标检测检测技术的学习，选用 YOLOv2 在实际开发平台 NVIDIA Jetson TX2 上开展实验与研究。针对 YOLOv2 和 TX2 的自身的体系结构特点，设计三种优化加速方案，分别是：

1. 调整参数及网络结构。通过参数的调整达到原始模型的最高性能，通过网络结构调整与压缩减少整个网络的计算量，从而加速整个网络的检测过程；

2. 采用半精度浮点数。在图片预处理之后，将 32 位浮点数转换为 half 类型之后再参与网络运算，从而减少了运算量以及数据传输量，以微小的准确度损失换得较大的速度提升；

3. 多线程 CPU-GPU 流水线设计。为充分利用 GPU 的计算能力，在 GPU 对当前输入帧进行计算的同时，进行 CPU 对下一帧的预处理操作，让 GPU 一直处于计算中，来实现系统的整体加速。

上述优化方法主要基于体系结构特点实现的，对不同的数据集和应用均可采用上述优化方法来进行加速设计，因此对多种应用场景有一定的通用性。

在实际应用中，往往要实现对运动物体的实时目标检测对其进行定位。根据检测目标的运动特点，在前三种方案的基础上，将改进的 YOLOv2 与目标追踪算法相结合，经过分析选择与 GOTURN 目标追踪相结合且速度得到进一步的提升。而最后基于平滑性的运动物体设计的位置预测算法的结合调度，使得检测速度达到 30fps 以上。

在之后的工作研究中，可以对 YOLOv2 网络的压缩进一步研究，探索采用量化技术节省网络的计算从而加速网络。另外，还可以开展采用 DVFS（动态电压频率调整）技术来降低系统能耗的优化研究。

# 参考文献

[1]Gauman K , Leibe B . Visual Object Recognition[C]// Annu Rev Neurosci. Morgan & Claypool Publishers, 2010.

[2]Strigl D , Kofler K , Podlipnig S . Performance and Scalability of GPU-Based Convolutional Neural Networks[C]// Euromicro International Conference on Parallel. IEEE, 2010.

[3]Sermanet P, Eigen D, Zhang X, et al. Overfeat: Integrated recognition, localization and detection using convolutional networks[J]. arXiv preprint arXiv:1312.6229, 2013.

[4]Liu L , Ouyang W , Wang X , et al. Deep Learning for Generic Object Detection: A Survey[J]. 2018.

[5]M. Everingham, L. Van Gool, C. K. Williams, J. Winn, andA. Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303–338, 2010.

[6]Whitehill J, Omlin C W. Haar features for facs au recognition[C]//7th International Conference on Automatic Face and Gesture Recognition (FGR06). IEEE, 2006: 5 pp.-101.

[7]Dalal, Navneet, Triggs, et al. Histograms of Oriented Gradients for Human Detection[C]// IEEE Computer Society Conference on Computer Vision & Pattern Recognition. 2005.

[8]Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2):91-110.

[9]Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[J]. CVPR (1), 2001, 1: 511-518.

[10]Felzenszwalb P, Mcallester D, Ramanan D. A discriminatively trained, multiscale,

deformable part model[C]// IEEE Conference on Computer Vision & Pattern Recognition. 2008.

[11]Hosang J, Benenson R, Dollár P, et al. What makes for effective detection proposals?[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 38(4): 814-830.

[12]Girshick R, Donahue J, Darrell T, et al. Region-based convolutional networks for accurate object detection and segmentation[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 38(1): 142-158.

[13]Girshick R. Fast r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.

[14]Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[C]//Advances in neural information processing systems. 2015: 91-99.

[15]He K, Zhang X, Ren S, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2015, 37(9): 1904-1916.

[16]Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[17]Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 7263-7271.

[18]Redmon J , Farhadi A . YOLOv3: An Incremental Improvement[J]. 2018.

[19]Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//European conference on computer vision. Springer, Cham, 2016: 21-37.

[20]Uijlings J R R, Van De Sande K E A, Gevers T, et al. Selective search for object recognition[J]. International journal of computer vision, 2013, 104(2): 154-171.

[21]Zitnick C L, Dollár P. Edge boxes: Locating object proposals from edges[C]//European conference on computer vision. Springer, Cham, 2014: 391-405.

[22]Held D , Thrun S , Savarese S . Learning to Track at 100 FPS with Deep Regression Networks[J]. 2016.

[23]Normalization B. Accelerating deep network training by reducing internal covariate shift[J]. CoRR.–2015.–Vol. abs/1502.03167.–URL: http://arxiv. org/abs/1502.03167, 2015.

[24]Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[25]Kirk D. NVIDIA CUDA software and GPU parallel computing architecture[C]//ISMM. 2007, 7: 103-104.

[26]Zhao R, Niu X, Wu Y, et al. Optimizing CNN-based object detection algorithms on embedded FPGA platforms[C]//International Symposium on Applied Reconfigurable Computing. Springer, Cham, 2017: 255-267.

[27]Qiu J, Wang J, Yao S, et al. Going deeper with embedded fpga platform for convolutional neural network[C]//Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016: 26-35.

[28]Mao H, Yao S, Tang T, et al. Towards real-time object detection on embedded systems[J]. IEEE Transactions on Emerging Topics in Computing, 2018, 6(3): 417-431.

[29]Tijtgat N, Van Ranst W, Goedeme T, et al. Embedded real-time object detection for a UAV warning system[C]//Proceedings of the IEEE International Conference on Computer Vision. 2017: 2110-2118.

[30]Smith H H. Object Detection and Distance Estimation Using Deep Learning Algorithms for Autonomous Robotic Navigation[J]. 2018.

[31]Wang C, Wang Y, Han Y, et al. CNN-based object detection solutions for embedded heterogeneous multicore SoCs[C]//2017 22nd Asia and South Pacific Design Automation Conference. IEEE, 2017: 105-110.

[32]Redmon, J.: Darknet: open source neural networks in c (2013–2016). http://pjreddie.com/darknet/

[33]Dundar A, Jin J, Martini B, et al. Embedded streaming deep neural networks accelerator with applications[J]. IEEE transactions on neural networks and learning systems, 2017, 28(7): 1572-1583.

[34]Chetlur S, Woolley C, Vandermersch P, et al. cudnn: Efficient primitives for deep learning[J]. arXiv preprint arXiv:1410.0759, 2014.

[35]Mittal S, Vetter J S. A survey of CPU-GPU heterogeneous computing techniques[J]. ACM Computing Surveys (CSUR), 2015, 47(4): 69.

[36]Nvidia C. Nvidia cuda c programming guide[J]. Nvidia Corporation, 2011, 120(18): 8.

[37]Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[38]Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

[39]Szegedy C , Liu W , Jia Y , et al. Going Deeper with Convolutions[C]// 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2015.

[40]He K , Zhang X , Ren S , et al. Deep Residual Learning for Image Recognition[C]// 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 2016.

[41]Howard A G , Zhu M , Chen B , et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications[J]. 2017.

[42]Zhang X , Zhou X , Lin M , et al. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices[J]. 2017.

[43]Szegedy C, Reed S, Erhan D, et al. Scalable, High-Quality Object Detection[J]. Computer Science, 2014.

[44]Russakovsky O, Deng J, Su H, et al. ImageNet Large Scale Visual Recognition Challenge[J]. International Journal of Computer Vision, 2015, 115(3):211-252.

# 致 谢

时间如白驹过隙，三年的硕士研究生学习生涯已经接近尾声，七年的山大时光即将结束。在论文即将完成之际，向所有给我关心和帮助的老师、同学和亲友们表示由衷的感谢。

感谢黎峰副教授和鞠雷副教授，两位老师无论在学术上还是在生活中都给与了我细心的指导和帮助，他们严谨的治学态度、精益求精的工作作风、诲人不倦的高尚师德时刻影响着我，两位恩师的谆谆教诲我将铭记在心。

感谢实验室的所有的小伙伴们：李世清、郝岳明、于淼、荣雅洁、林茂、秦晓宇、薛鹏飞、麻乔妮和赵彦博，感谢他们陪我度过了难忘的科研时光，还要特别感谢已经毕业的师兄师姐们：臧传奇、徐玉景、李涵、崔博、隋晓金，感谢他们在我刚来实验室时给与的耐心指导和帮助。

特别感谢我的父母在我多年的求学道路上给予的支持和鼓励，在未来的日子里，我会更加努力的学习和工作！

最后，再次向所有关心我的亲人、师长和朋友们表示深深的谢意。在即将离校之际，祝愿所有老师和同学们健康快乐！

柳佳园

2019.3.15

# 学位论文评阅及答辩情况表

| | 姓　名 | 专业技术职　务 | 是否博导（硕导） | 所在单位 | 总体评价※ |
|---|---|---|---|---|---|
| 论文评阅人 | 匿名评阅人1 | - | - | - | |
| | 匿名评阅人2 | - | - | - | |
| | 匿名评阅人3 | - | - | - | |
| | | | | | |
| | | | | | |

| | | 姓　名 | 专业技术职　务 | 是否博导（硕导） | 所在单位 |
|---|---|---|---|---|---|
| 答辩委员会成员 | 主席 | 张化祥 | 教授 | 博导 | 山东师范大学 |
| | 委员 | 许信顺 | 教授 | 博导 | 山东大学 |
| | | 刘卫国 | 教授 | 博导 | 山东大学 |
| | | 王华 | 教授 | 博导 | 山东大学 |
| | | 李新 | 副教授 | 硕导 | 山东大学 |
| | | 鞠雷 | 副教授 | 硕导 | 山东大学 |
| | | | | | |
| | | | | | |
| | | | | | |

| 答辩委员会对论文的总体评价※ | B | 答辩秘书 | 李巍 | 答辩日期 | 2019.5.15 |
|---|---|---|---|---|---|
| 备注 | | | | | |

※优秀为“A”；良好为“B”；合格为“C”；不合格为“D”。