中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

基于国产化 AI 芯片的目标检测研究

作者姓名：杨嘉棋

指导教师：欧阳益民 高级工程师 中国科学院光电技术研究所

学位类别：工学硕士

学科专业：信号与信息处理

培养单位：中国科学院光电技术研究所

2021 年 6 月

Research on Target Detection Based on Native AI Chip

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Science in Engineering

in Signal and Information Processing

By

Yang Jiaqi

Supervisor: Senior Engineer OuYang YiMin

The Institute of Optics and Electronics,

The Chinese Academy of Sciences

June 2021

中国科学院大学

研究生学位论文原创性声明

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工作所取得的成果。尽我所知，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研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：杨嘉模

日　　期：2021.5.25

中国科学院大学

学位论文授权使用声明

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：杨嘉模　　导师签名：

日　　期：2021.5.25　　日　　期：2021.5.25

摘 要

近年来，伴随着深度神经网络特征学习能力的不断提升，采用深度学习技术的目标检测算法在计算复杂度上越来越高，由此催生出了专用的 AI 计算芯片。如 Nvidia 的 GPU、Google 的 TPU 等相继涌现，它们具有较强的计算能力和良好的开发环境，可便捷地进行相应项目的开发。然而，在当前国际局势复杂多变，国外先进技术对我国逐步封锁和关键领域“卡脖子”的情况下，基于国外 AI 芯片开发的目标检测应用，在安全性、可持续性和稳定性等方面都存在不可预知的隐患。因此，开展基于国产 AI 芯片的目标检测算法研究，探索推进基于国产芯片 AI 项目的应用，意义重大且迫在眉睫。

1. 本文通过国内 AI 芯片的梳理和对比分析，从性能、功耗和扩展性等方面，以海思的 Hi3559AV100 芯片为基础，进行了平台的总体设计与部署、系统配置与管理、开发环境搭建与集成等研究工作，形成了一套通用的基于该国产芯片的自主可控、稳定可靠和安全高效的智能化系统平台。

2. 在基于该国产 AI 芯片的智能处理平台上，对基于深度神经网络的目标检测算法进行了部署研究。对比选取了两阶段网络模型中典型的 R-FCN 和单阶段网络模型中的 YOLOv3，并根据其各自的算法特点与部署实现的需求，在该智能平台的有限资源下，进行了算法优化、算法模型与开发环境协同的模型转换、算法与平台资源的协同设计等研究。在设计的单芯片并行处理架构下，提高了目标检测算法在智能处理平台上的检测帧率。最后，通过公开的数据集，对部署后的两种网络模型进行了对比测试，验证了基于国产 AI 芯片所开发智能处理平台的有效性与通用适应性。

3. 针对智能处理平台在实际应用中的可靠性和实时性，进行了验证和优化。在具体的无人机检测项目中，通过结合自建的无人机数据集和 Hi3559AV100 中神经网络推理的专用硬件单元 NNIE，采用聚类和集成剪枝的方法，对选用的 YOLOv3-tiny 网络进行了优化，在保证算法精度的同时，缩减了模型的尺寸大小，提高了网络在智能平台上的推理速度。最终，在实际的项目中实现了无人机的目标检测，满足了项目的应用需求。

综上，针对 AI 技术日益广泛的应用，其大量数据处理能力的需求对处理平台提出了极高的要求，因而很大程度地依赖于国外芯片平台，但由于国外技术的封锁与“卡脖子”使得国外芯片平台风险很大，对此进行了基于国产芯片智能处理平台的研究、智能平台目标检测算法部署的研究及应用项目的研究，形成了相应的智能处理平台与算法部署方案，通过相关实验验证了基于国产芯片进行智能技术研究应用的有效性。

关键词：国产 AI 芯片，智能处理平台，目标检测，Hi3559AV100

Abstract

In recent years, with the continuous improvement of deep neural network feature learning capabilities, target detection algorithms using deep learning technology have become more and more computationally complex, which has given birth to dedicated AI computing chips. For example, Nvidia's GPU, Google's TPU, etc. have emerged one after another. They have strong computing power and a good development environment, which can facilitate the development of corresponding projects. However, in the current complex and changeable international situation, foreign advanced technology is gradually blocking my country and key areas are "stuck", the target detection application based on foreign AI chips is developed in terms of safety, sustainability, and stability. There are unpredictable hidden dangers. Therefore, it is of great significance and imminent to carry out research on target detection algorithms based on domestic AI chips, and to explore and promote the application of AI projects based on domestic chips.

1. Through combing and comparative analysis of domestic AI chips, in terms of performance, power consumption and scalability, based on HiSilicon's Hi3559AV100 chip, this paper has carried out the overall design and deployment of the platform, system configuration and management, and development environment construction. Integration and other research work has formed a universal intelligent system platform based on this domestic chip that is autonomous, controllable, stable, reliable, safe and efficient.

2. On the intelligent processing platform based on the domestic AI chip, a research on the deployment of a target detection algorithm based on a deep neural network was carried out. The typical R-FCN in the two-stage network model and the YOLOv3 in the single-stage network model were selected for comparison, and according to their respective algorithm characteristics and deployment requirements, algorithm optimization and algorithm were carried out under the limited resources of the intelligent platform. Research on model conversion between model and development environment, and collaborative design of algorithm and platform resources. Under the proposed single-chip parallel processing architecture, the detection frame rate of the target detection algorithm on the intelligent processing platform is improved; finally, through the public data set, the two network models after deployment are compared and

tested, verifying that the target detection algorithm is based on domestic The effectiveness and general adaptability of the intelligent processing platform developed by the AI chip.

3. The reliability and real-time performance of the intelligent processing platform in practical applications have been verified and optimized. In the specific UAV detection project, by combining the self-built UAV data set and the special hardware unit NNIE for neural network inference in Hi3559AV100, clustering and integrated pruning methods are used to perform the selected YOLOv3-tiny network. The optimization, while ensuring the accuracy of the algorithm, reduces the size of the model, and improves the inference speed of the network on the intelligent platform. In the end, the target detection of the UAV was realized in the actual project, which met the application requirements of the project.

To sum up, in view of the increasingly widespread application of AI technology, the demand for its massive data processing capabilities has placed extremely high requirements on the processing platform, and thus largely relies on foreign chip platforms, but due to the blockade and "stuck neck" of foreign technology This makes foreign chip platforms very risky. In this regard, researches based on domestic chip intelligent processing platforms, intelligent platform target detection algorithm deployment research and application projects have been carried out, and the corresponding intelligent processing platform and algorithm deployment plan have been formed, and related experiments have been carried out. Verifies the effectiveness of intelligent technology research and application based on domestic chips.

Key words: Domestic AI chip, Intelligent processing platform, Target detection, Hi3559AV100

目 录

图目录

表目录

第 1 章　绪论

1.1　研究背景与意义

计算机视觉作为一门交叉学科，研究如何通过计算机系统从现实环境中的数字图像或视频中提取高层次信息 [1]，与其相关的技术不断地取得了创新和发展，并在人们的日常生活中扮演了越来越重要的角色。目标检测作为计算机视觉领域活跃的研究方向之一，主要任务是解决目标图像中存在什么样的物体，在哪里的问题 [2]。目前，目标检测在智能视频监控、机器人导航、身份识别和道路检测等领域都得到了实际的部署和应用。

近年来，随着深度学习技术的不断创新和发展，其在目标检测算法上得到了广泛的应用 [3]。然而，基于深度神经网络的目标检测算法，在实际的应用过程中，存在大量的矩阵乘加运算操作，整体的计算量越来越大，对计算资源并行化程度的需求也越来较高，传统的计算架构在计算速度和功耗方面都已无法满足深度学习算法在实际工程落地过程中的需求 [4]。因此，面向深度学习算法的专用人工智能芯片应运而生，该类芯片的出现，极大地提高了算法的运行效率，有效地促进了整个行业的发展。除了在芯片方面，设计研发专用的人工智能硬件来弥补现有设备计算资源的不足，还可针对神经网络本身，采用剪枝和量化的方法，对已有的算法模型进行压缩和优化，以达到缩减网络模型计算复杂度和尺寸大小的目的，从而可以更加高效地利用已有的硬件资源，扩大基于深度学习的目标检测算法在移动平台上的应用。

目前，国外芯片在基于深度学习的目标检测算法领域占据了大部分的市场份额，许多主流的 AI 应用也依赖国外的芯片进行设计和开发。然而，随着国际局势的复杂多变和我国综合国力的不断提升，在我国向世界强国不断发展前进的道路上，国外先进技术开始逐步对我国进行封锁，在相应的关键领域，对我国进行“卡脖子”；除此之外，国内应用厂商对国产化芯片的需求也开始逐渐增加，尤其是在某些特定的行业和领域，对器件的国产化要求也越来越严格。因此，非常有必要基于国产自有知识产权的 AI 芯片，进行目标检测算法的研究，探索推进基于国产芯片的 AI 项目应用。

1.2 国内外研究现状

自人工智能技术提出以来，有关的研究经历了两起两落的跨越式发展[5]。随着 AlexNet[6]在 2012 年度的 ImageNet 比赛中取得优异的成绩和阿尔法狗[7]战胜世界第一的围棋高手，人工智能技术再次进入了迅猛的发展阶段。多个国家也都从国家战略层面，出台了有关人工智能的发展规划，并建立了相对完整的研究发展机制[8][9]。其中，作为深度学习算法部署运行的实际载体和人工智能行业底层架构的芯片产业，更是得到了重点的关注和发展。

目前，在芯片领域，对 AI 芯片并没有一个标准的定义，普遍认为针对深度神经网络和人工智能应用进行特殊加速的芯片都可称之为 AI 芯片[10]。因此，如图 1.1 所示，AI 芯片从不同的方面可以被划分为不同的类别。

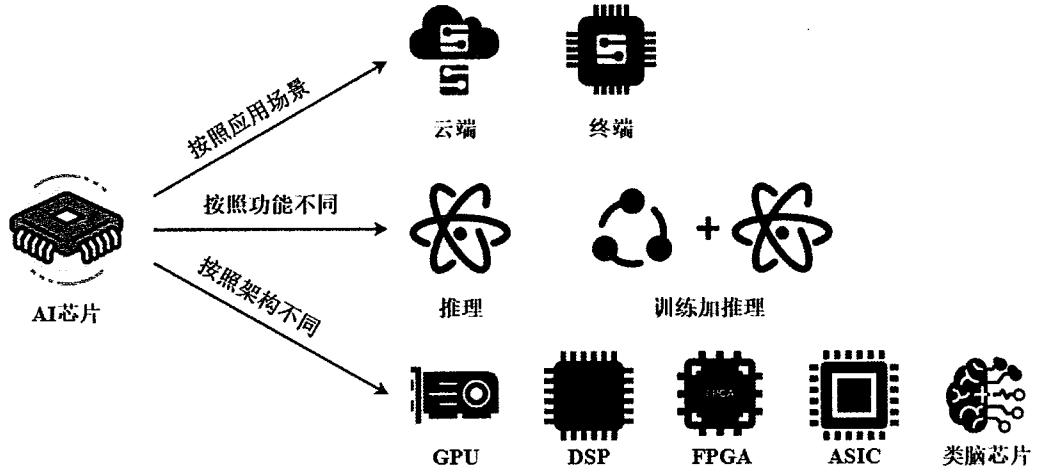


图 1.1 AI 芯从不同方面的类别划分

Figure1.1 AI chips are divided into different categories

按照应用场景的不同，可以划分为云端和终端芯片；按照功能的不同，可以划分为专门面向推理加速的芯片和兼具训练与推断两方面功能的处理芯片；按照采用的技术架构，还可分为 GPU（graphics processing unit）、DSP（digital signal processing）、FPGA（field-programmable gate array）、ASIC（application specific integrated circuits）和神经拟态类脑芯片等[5]。在功能的划分中，具备两方面功能的芯片，一般以集群的形式部署在云端，整体性能强劲，但是功耗较高，且需要通讯网络的支撑，在某些应用方面，会产生一系列网络延迟和带宽的问题，对于某些特定的领域，实现上也存在一定的限制，例如，将大型城市数百万及以上高清摄像头的人脸识别算法，全部托管到云端设备上进行运行和检测，大量的实时

信息传输会让整个通信系统难以正常运行；而仅面向推理的加速芯片，主要散布在移动终端，针对性强，芯片所需的算法模型在 PC 训练完成后可移植到硬件平台上，在离线模式下运行，不受网络的限制，具备成本低廉、高效、紧凑和低功耗的特点，符合未来的应用需求。

在架构的划分中，GPU 最早是出于图像渲染的需要而设计的芯片，具备矩阵运算能力，善于执行并行计算，最先应用在深度学习算法的加速过程中，但相比专门的 AI 芯片，存在能耗大和效率低的问题；DSP 作为处理各种信号信息的处理器，适配深度学习算法需要针对特定的 IP 核，能够处理的任务较为单一；FPGA 作为可编程逻辑器件，可实现硬件层的编程和配置，对使用者的要求较高，开发难度较大，相比于专用芯片，在深度学习算法的应用性能和功耗方面存在一定的差距；神经拟态类脑芯片作为模拟生物脑的芯片，可在极低的功耗下运行，通过类似脑神经的方式进行运算，但该类芯片尚处于研究阶段，不适用于目前主流的神经网络拓朴结构，而作为专用集成电路的 ASIC 芯片，具备定制化能力，在目前深度学习算法的应用过程中，在芯片体积、性能、功耗和可靠性等方面都要优于同类芯片，并在近年来，得到了大多数公司的青睐 [11][4]。因此，综合 AI 芯片的应用场景、功能和技术架构，针对终端推理的专用 ASIC 芯片，更利于实际的工程开发和应用。

当前，越来越多的组织和公司开始投入到 AI 芯片领域的探索和研发中。比如国外的谷歌、英特尔和亚马逊等大型公司，都在该方面进行了研究，在这些企业之中，以谷歌的 TPU 最具代表性，其在速度和能效方面，相比同期的 GPU，均有较大提升，其中，平均提速可达 15~30 陪，能效比提高至 30~80 倍 [5]。在国内，如表 1.1 所示，寒武纪、比特大陆和瑞芯微等众多技术公司，也都相应的推出了面向终端移动场景的神经网络加速芯片。

表 1.1　国内部分终端和边缘 AI 芯片 [12]

Table1.1　Some domestic terminals and edge AI chips

芯片名称	算力（TOPS）	功耗	工艺	特性
瑞芯微 RK3399Pro	3T	10W	28nm	具备多终端互联和编解码的功能
瑞芯微 RK1808	3T	3W	22nm	形态灵活、可扩展性强；兼容多种深度学习算法和网络模型。

云天励飞 DeepEye1000	2T	2W	22nm	自主可编程，基于自研的NNP和ASIP指令集，支持级联扩展。
寒武纪思元 220 M.2	8T	8.25W	16nm	计算可编程，采用自研的MLUv02架构。
比特大陆 BM1880	1T	2W	28nm	采用特殊设计的调度引擎，可为核心处理器提供较高的数据流，支持多种AI编程框架。
海思 Hi3519AV100	2T	1.9W	12nm	提供多种图像校正和增强算法，支持多路IPC接入和图像编解码。
海思 Hi3559AV100	4T	3W	12nm	集成了NNIE、DSP等智能处理器，采用了低功耗架构和工艺，具备出色的图像处理能力，拥有丰富的计算资源。

从表格 1.1 中可以得出以下几点信息：

1. 在算力方面，国产的寒武纪思元 220 M.2 最为突出，高达 8 TOPS（Tera Operations Per Second），次之为海思的 Hi3559AV100、瑞芯微的 RK3399Pro 和 RK1808，整体的芯片算力都在 3T 及以上；

2. 在功耗方面，除了瑞芯微的 RK3399Pro 和寒武纪思元 220 的功耗较高外，其余芯片的典型功耗均在 3W 及以下，突出了专用芯片低功耗的特点；

3. 算力较高的几款芯片中，其典型功耗对应下的计算能耗比分别约为 0.97 TOPS/W、1.33 TOPS/W、1 TOPS/W 和 0.3 TOPS/W，其中能耗比相对较高的是海思的 Hi3559AV100 和瑞芯微的 RK1808，能耗比均在 1 TOPS/W 及以上。

因此，综合表格已有信息，海思的 Hi3559AV100 在算力、功耗和制程工艺上都能够做到有效的均衡。在功能能够满足大部分人工智能应用需求的同时，该芯片还拥有丰富的图像处理能力和强劲的图像视频编解码功能，并提供了多样化的视频输入输出和开发接口，有利于后期的二次开发和应用拓展。

目前，人工智能产业正处于高速的发展中。在人工智能芯片的应用方面，各领域的智能算法和架构正位于快速迭代的发展阶段，除了芯片本身的高性能、低功耗和强适应性外，考虑到芯片较长的生产周期和较高的研发成本，还需要 AI 芯片企业具备成熟配套的场景解决方案，广阔的业务应用场景，稳定可靠的开发

环境和雄厚的经济实力。知名市场调研咨询公司 Compass Intelligence 从供应商指标、产品和客户指标以及经济指标等其它方面，对全球范围内 100 多家 AI 软硬件芯片公司进行了调研和排名[13]，2019 年度的最新结果显示，位列前十位的芯片公司分别为英伟达（Nvidia）、英特尔、恩智浦（NXP）、苹果（Apple）、谷歌、AMD、华为、Imagination、ARM/Softbank 和高通（Qualcomm）。可见华为在目前的 AI 芯片领域处于领先地位，具备广阔的发展前景和市场。

目标检测作为一种与计算机技术相关的检测方法，主要用于定位和识别给定数字图像或视频中的目标对象[14]。过去几十年，目标检测算法在人脸和文本识别，以及行人和遥感检测等领域得到了广泛的应用。其中，作为最早应用方向之一的人脸识别，在各行各业已经得到了实际的部署，例如各种支付平台的人脸认证、数码相机中的微笑检测、手机等各种移动设备的人脸解锁以及移动程序中的面部美化等等；除此之外，行人检测作为另外一个重要的应用方向，在自动驾驶、刑事案件侦查和智能视频监控等许多领域都得到了广泛的部署[15]。综上所述，不管是人脸识别在数码相机中的微笑检测，还是行人检测在安防领域的部署，都体现了边缘端设备对目标检测算法进行处理的应用趋势[16]。因此，本文综合以上信息和目前 AI 芯片的国内外发展现状，以及国产化的未来趋势，从实验室视觉信息处理为主的需求出发，选用华为海思 Hi3559AV100 的芯片平台进行研究。

1.3　本文主要研究内容和组织结构

1.3.1　研究内容

本文基于国产自有知识产权的 AI 芯片，对目标检测算法进行了研究。通过对国内不同功能与特征 AI 芯片的分析和对比，以国产的 Hi3559AV100 芯片为基础，进行了智能平台的总体设计与部署、系统配置与管理、开发环境搭建与集成等研究工作。根据目标检测算法的特点与算法部署实现的需求，在智能平台的有限资源下，对基于深度学习的两阶段算法模型 R-FCN[17]和单阶段算法模型 YOLOv3[18]，进行了算法优化、算法模型与开发环境协同的模型转换、算法与平台资源的协同设计等研究，并通过算法的仿真实验，实现了算法模型在智能平台上的最终部署，验证了基于国产 AI 芯片所开发智能处理平台的有效性与通用适应性。针对具体的无人机检测项目，在选用的 YOLOv3-tiny[19]算法基础上，根据采集的无人机数据集，通过聚类算法对网络的锚点框进行了重新适配，并针对

Hi3559AV100 中神经网络推理的 NNIE 硬件单元，采用集成剪枝的方法，对 YOLOv3-tiny 的网络结构进行了优化，在保证网络精度的同时，缩减了模型的尺寸大小，提高了网络在智能平台上的推理速度，满足了实际的应用需求，验证了平台在实际项目应用中的可靠型，从而形成了一套通用的基于该国产芯片的拥有自主知识产权、安全可靠和精简高效的智能处理平台

1.3.2 组织结构

本文整体的组织结构如下：

第一章为绪论，包括了论文研究的背景和意义，同时梳理了国内外 AI 芯片的发展现状，并对本文的研究内容和组织结构进行了安排。

第二章主要对智能平台和卷积神经网络的相关技术背景进行了介绍。智能平台部分主要包括核心芯片的基本参数和主要功能，平台内部整体的媒体控制流程和处理方式，以及核心的智能视觉异构加速平台。卷积神经网络部分则主要对神经网络的基本原理和基础结构进行了简单阐述。

第三章则主要围绕智能处理平台的总体设计和国产芯片集成开发环境的搭建进行展开。首先，对平台的总体处理流程进行了设计，并具体阐述了对应流程下的硬件模块；之后，配置和搭建了所需的集成开发环境，并在该环境基础上，移植烧录了平台系统所需的 U-boot、Linux 内核和 ext4[20]文件系统，对国产智能处理平台的软硬模块进行了适配。

第四章主要在国产的智能处理平台下，对基于深度学习的目标检测算法进行了部署研究。在本章中，首先对目标检测的传统方法和基于深度神经网络的方法进行了简单的对比和介绍，之后比较选取了两阶段的 R-FCN 和单阶段的 YOLOv3 算法，接着对其各自的算法原理、仿真验证、网络模型的协同转换和平台有限资源下的协同设计部署进行了详细的阐述，最后在公开数据集上分别对部署后的算法模型进行了对比测试。

第五章主要是有关国产智能平台在目标检测方面的应用。首先介绍了无人机目标检测的实际应用需求，在 YOLOv3 和 YOLOv3-tiny 的网络模型中，对比选择了 YOLOv3-tiny 作为最终的部署算法，接下来，在原有 YOLOv3-tiny 网络的基础上，通过聚类和剪枝的方法，对其进行了进一步的优化，并对比了优化前后

的检测效果和推理速度，之后通过模型转换和算法仿真对网络模型进行了验证和测试，验证测试通过后，将优化后的模型在国产智能平台上进行了部署实现。

第六章是全文的总结和展望，概括了本文整体的研究内容，并根据当前的不足，阐明了未来的发展方向。

第 2 章　智能平台和神经网络相关理论

2.1　智能平台简介

2.1.1　海思 Hi3559 芯片

本文采用的国产化 AI 芯片为海思推出的 Hi3559AV100。该芯片集成了双核的 A73 和双核的 A53，具备智能视觉处理和出色的视频及图形处理能力；集成了高性能的 ISP 模块，拥有丰富的计算资源，支持多路视频输入和视频编解码，可实现多路图像处理和影视级 RAW 数据输出的功能。此外，还拥有多种智能加速引擎，其中包括神经网络推理加速引擎 NNIE（Neural Network Inference Engine），该引擎是 CNN、RCNN 等神经网络结构的深度学习算法主要使用到的专用硬件单元，支持卷积、Pooling 和 Pad 等相关的操作，能够提供 4TOPS 的算力，可以在图片分类、目标检测等场景得到应用。整个芯片在设计过程中还采用了大小核和专为低功耗所设计的硬件架构，在性能和功耗方面可以做到有效的均衡。芯片整体的功能框图如图 2.1 所示 [21]。

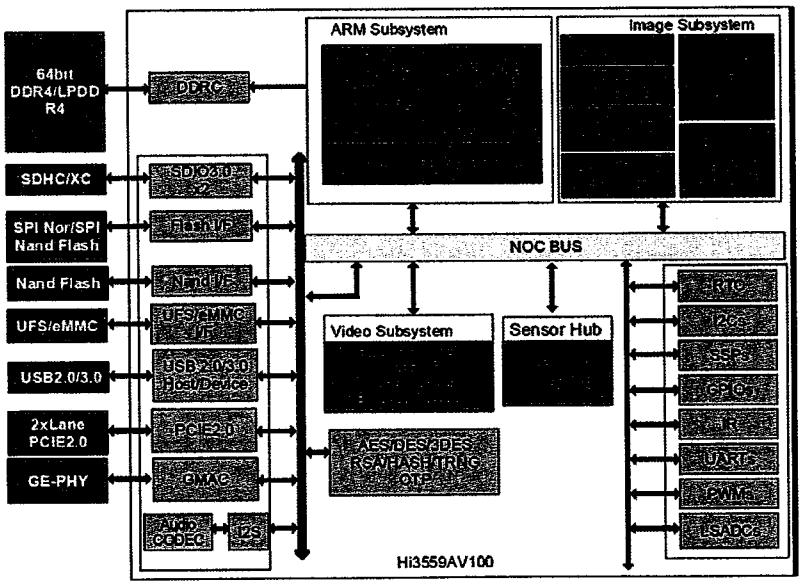


图 2.1　Hi3559AV100 功能框图

Figure2.1　The block diagram of Hi3559AV100 functional

除了 NNIE 之外，该芯片内部还包含了其它的智能加速引擎，包括视觉处理加速的 Vision DSP、计算深度图的 DPU（Depth Process Unit）和智能视频引擎

IVE（Intelligent Video Engine）。其中，Vision DSP 作为专用的视觉处理加速器，支持标量定点、矢量定点和浮点运算，可提供智能分析算法所需的计算资源，并编程实现相应的算法；DPU 可通过对输入图像的校正和匹配，计算出深度图，能够实现亚像素深度图的输出；IVE 是智能分析系统中的硬件加速单元，能够提供智能分析算法所需要的基础运算功能和执行一些耗时较大的操作，支持图像的颜色空间转换、LK 光流运动估计和角点检测等，极大的提高了整个芯片的运行效率。

芯片主控部分，主要由双核的 ARM Cortex A73 和双核的 ARM Cotex A53 组成，两者均采用非对称多核架构，可实现统一的中断处理。其中，A73 主频最高可达 1.6GHz，支持电压频率的动态调整，具备 32KB 和 64KB 的指令和数据缓存，以及 512KB 的 L2 缓存，A53 具备 A73 相同大小的指令 Cache，数据 Cache 为 32KB，L2 Cache 为 256KB，主频最高可达 1.2GHz，同样支持电压频率的动态调整。双核 A73 和双核 A53 组和作为系统总体调度运行的同时，还可对 NNIE 推理得到的结果进行进一步的处理。

在芯片内部的图形和视频处理部分，含有利用硬件单独进行图形绘制的二维图形加速引擎 TDE（Two Dimensional Engine），能够实现图形的拷贝、填充、旋转等功能，同时，包括可执行视频缩放、幅型比转换、视频裁剪和解压缩等操作的视频处理子系统 VPSS（Video Processing Sub System），以及能够实现视频和图形共同处理的视频图形系统 VGS（Video Graphics System）等。在视频接口方面，输入视频的最大分辨率支持到 7680 x 4320，支持 8 路 sensor 数据的串行输入，支持 MIPI、LVDS、HiSPi、SLVS-EC 和 BT.1120 等视频输入接口和 MIPI DSI、HDMI 等输出接口。

在视频编解码方面，主要由 VEDU、JPGE、VDH 和 JPGD 四部分组成。其中，VEDU 和 VDH 分别对视频数据进行编码和解码，可针对感兴趣区域编码；JPGE 和 JPGD 则主要对图片进行 MJPEG 格式的编码和解码，编码过程中，可自行配置量化表，支持 32 x 32 到 16384 x 16384 分辨率范围内的大小。在外部接口方面，提供了 I^2C、USB 2.0/USB 3.0、UART、SD 3.0/SDIO 3.0 和 GMAC 以太网等接口。通过动态存储控制器 DDRC 接口，可实现 DDR4/LPDDR4 存储器的存取控制，该接口最大支持容量为 8GB，最高工作频率可达 1333MHz。通过 eMMC、

UFS 接口，可满足大容量存储的需求，其中，eMMC 5.1 接口最大可支持 2TB 的外部存储 [21][22]。

2.1.2　海思媒体处理平台

在通过 Hi3559AV100 实现目标检测算法的过程中，涉及到图像和视频数据流的控制与处理。为了实现媒体信息的高效控制，提高系统平台的研发效率，海思开发了专用的媒体处理软件平台 MPP（Media Process Platform）。在 MPP 内部，对不同的处理功能进行了模块化封装，可通过 MPP 平台提供的软件接口 MPI（MPP Program Interface），实现不同模块单元的控制，省去了调用芯片底层处理的复杂流程，达到了简洁、高效 [23]。MPP 平台的控制流程如图 2.2 所示，从平台应用层出发，根据需要调用对应的 MPI 接口，通过该接口到达操作系统的适配层，并通过该层的系统调用函数完成系统级的操作，最终达到控制底层芯片和外围器件的目的。如此的流程控制和层次划分，可使得整个平台兼容不同版本的同种操作系统，屏蔽不同操作系统的差异，大大提高了整体的适应性。

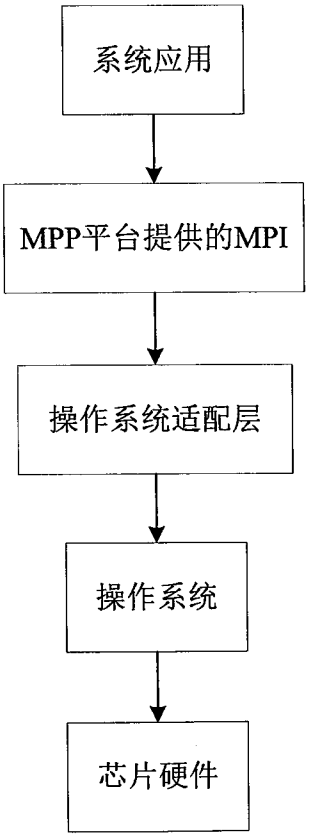


图 2.2　MPP 平台控制流程图

Figure2.2　The flow chart of the MPP platform controlling

MPP 内部主要由负责视频输入、处理和输出的 VI、VPSS、VO，以及负责区域管理的 REGION 模块组成，主要的处理流程如图 2.3 所示。

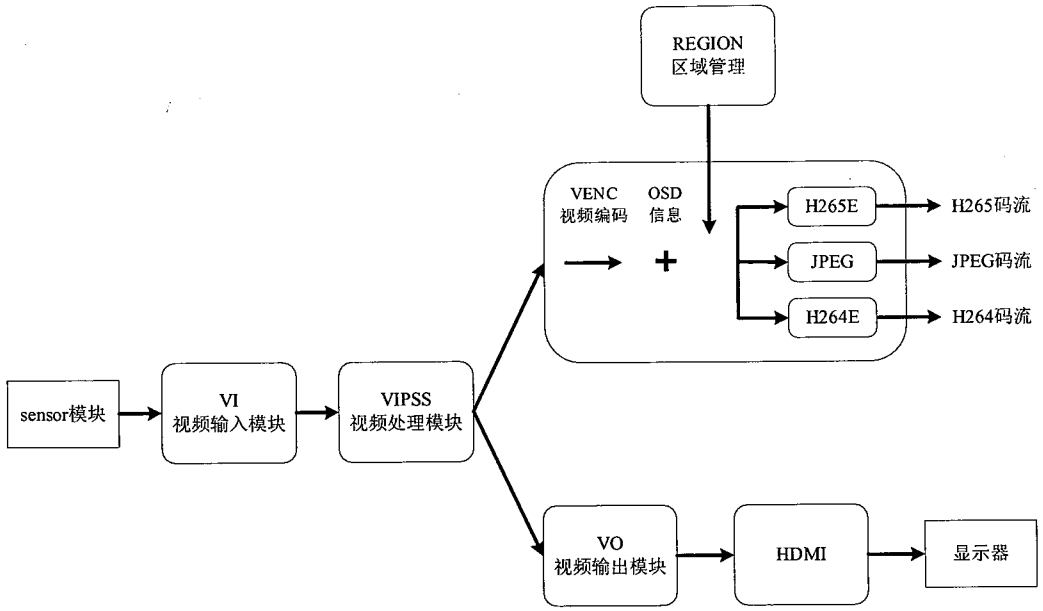


图 2.3　MPP 的主要处理流程

Figure2.3　Main processing flow of Media Process Platform

视频输入模块通过 sensor 捕获视频的图像数据流，并对捕获到的数据流进行裁剪、去噪等预处理，并将预处理后的图像数据发送给 VPSS 模块，VPSS 则在 VI 的基础上，对图像进行图像增强、去噪、锐化等操作。经 VPSS 处理后，传送给 VENC 模块的图像数据，可与 Region 模块设置的 OSD 图像进行叠加，并按照不同的协议进行编码输出；当通过 VO 模块对 VPSS 处理后的图像信息进行捕获时，则可实现播放控制和输出协议的自定义配置。

2.1.3　视觉异构加速平台

在海思芯片内部，提供了专门对图像、视频等视觉信息进行处理的智能视觉异构加速平台 SVP（Smart Vision Platform）。整个平台主要由硬件处理单元和硬件单元所需的开发环境两部分组成，开发框架如图 2.4 所示。从需要实现的应用需求出发，根据 SVP 提供的处理接口，配置选择需要的功能；被调用的接口通过链接内核中对应的驱动程序，实现特定硬件单元的控制。

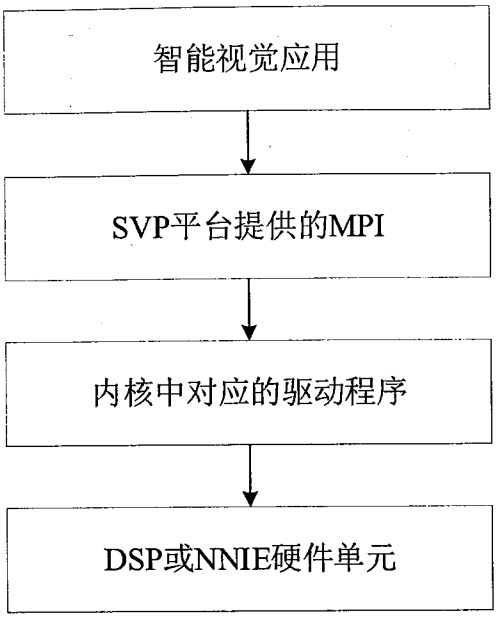


图 2.4　SVP 平台开发框架

Figure2.4　SVP platform development framework

在该视觉异构加速平台中，本文主要使用到 NNIE 硬件单元，其作为专门针对神经网络特别是卷积神经网络进行加速的模块，可支持目前大多数的公开算法模型，例如 VGG16[24]、Googlenet[25]、Alexnet[6]和 ResNet[26]等网络。当前，NNIE 配套的工具链仅支持以 Caffe[27]框架为基础实现的算法模型。

以基于 Caffe 框架的模型为例，整体的开发流程如图 2.5 所示。首先，在 PC 机上针对所需要的算法进行训练，得到对应的 Caffe 模型；之后，通过 NNIE 工具链中的 mapper 工具，将训练完成后得到的*.caffemodel 文件和*.prototxt 文件，转换为海思 NNIE 上能够加载执行的*.wk 数据指令文件；最后，通过编写相应的代码，将转换后的指令文件移植到目标平台，实现目标神经网络算法在 NNIE 硬件上的加速推理。其中，在 mapper 工具转换的过程中，可通过设置不同的转换模式，将训练后的模型转化为模拟仿真或板端推理两种不同的模型；模拟仿真模型可通过仿真库在 PC 端上加载执行，能够实现转换后模型在精度和宽带等方面的初步评估，该评估结果可反映出模型的优劣，利于使用者的前期开发；在评估符合目标需求时，可通过 mapper 工具将最终的网络算法转换为板端推理的模型，该类模型可移植到目标平台，由 NNIE 硬件单元直接加载执行。

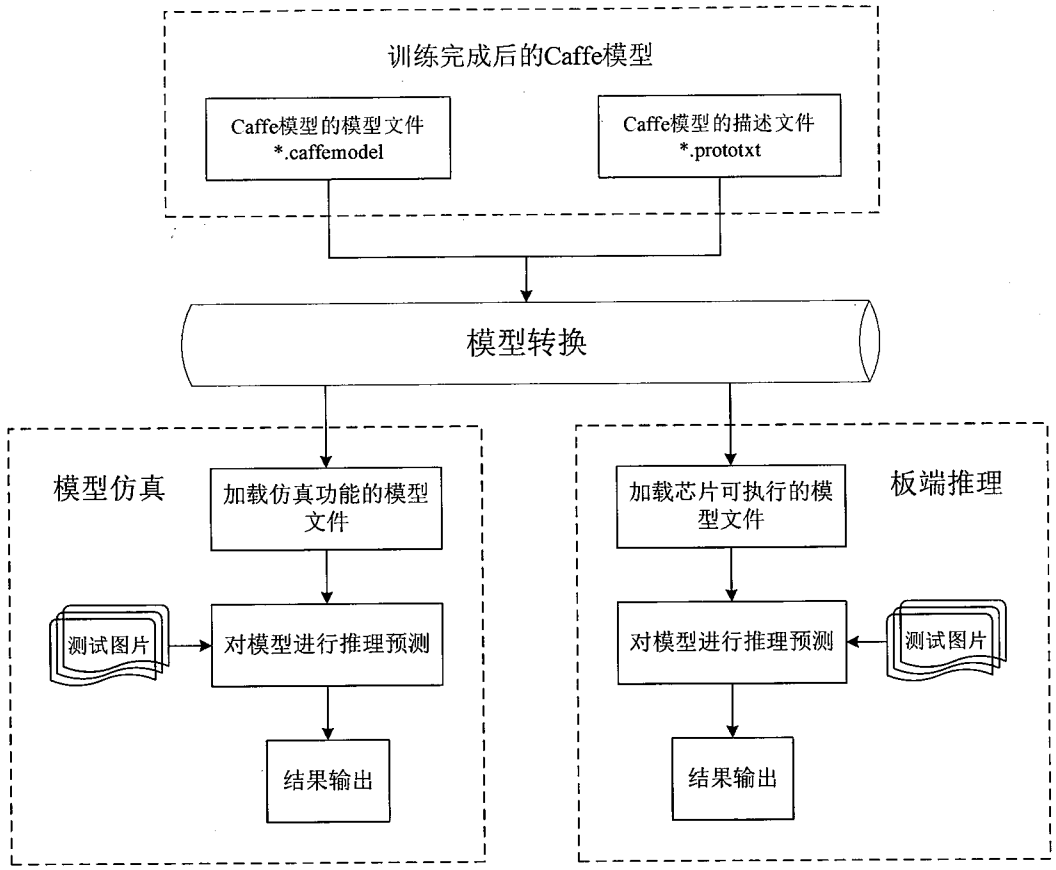


图 2.5　NNIE 开发流程

Figure2.5　NNIE development process

在 NNIE 硬件单元中，深度神经网络的网络层被划分为标准层、扩展层和 Non-support 层三类，标准层为 NNIE 能够运行的 Caffe 标准层，例如常见的卷积、池化和 Concat 等网络层，扩展层为 NNIE 支持，但 Caffe 模型框架不支持的网络层，例如 SSD[28]中使用到的 Normalize 网络层结构、YOLOv2[29]在 Darknet 模型框架下自定义的 Passthrough 层和 SegNet[30]中用于执行池化逆操作的 Upsample 层等，Non-support 层为 NNIE 不能够识别的网络结构，例如在 Caffe 中用于训练的层和 YOLO 算法模型中的 yolo 层。其中，为了使 NNIE 能够正常运行神经网络中的扩展层，需要在 Caffe 源码的 caffe.proto 上，对使用到的扩展层进行扩展，并增加对应的底层实现代码 [31]。

2.2　卷积神经网络原理

2.2.1　神经网络与激活函数

神经网络作为一种受到生物脑神经原理启发的计算结构，在大量的数据下，经过有监督的训练，可以习得训练样本的一般化特征，成为一个具备较强泛化性的神经网络模型。

神经网络主要由多层神经元组成，不同层的神经元承担着不同的角色，如图 2.6 所示。每一层的神经元都与其后的下一层神经元进行连接，通过增加隐含层的数量，形成深度神经网络，可有效提高网络模型的特征提取能力。

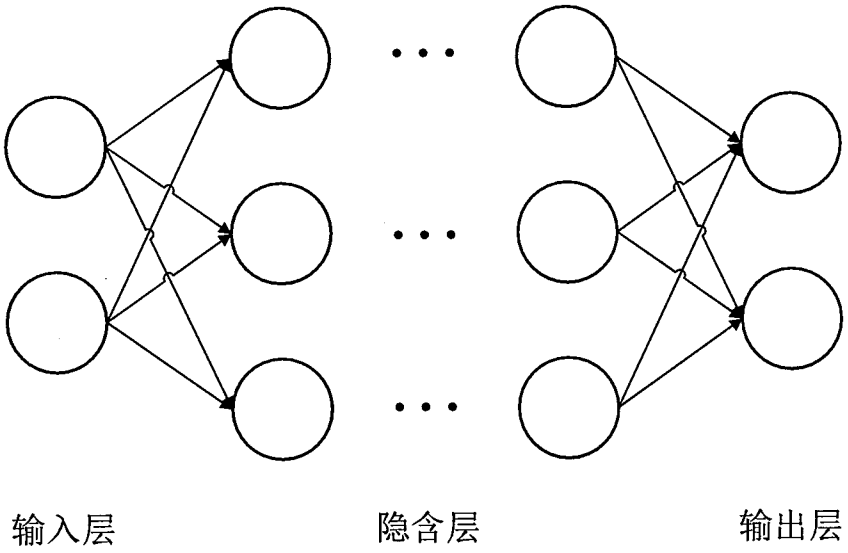


图 2.6　神经网络结构示意图

Figure2.6　Schematic diagram of neural network structure

在该类拓扑结构中，神经元会根据其对应的输入产生一个特定的输出。如图 2.7 所示，多个 x_i 为神经元的输入，w_i 为每个输入对应的权重，在将所有的输入加权求和后，通过激活函数 f 即可得到输出 y。

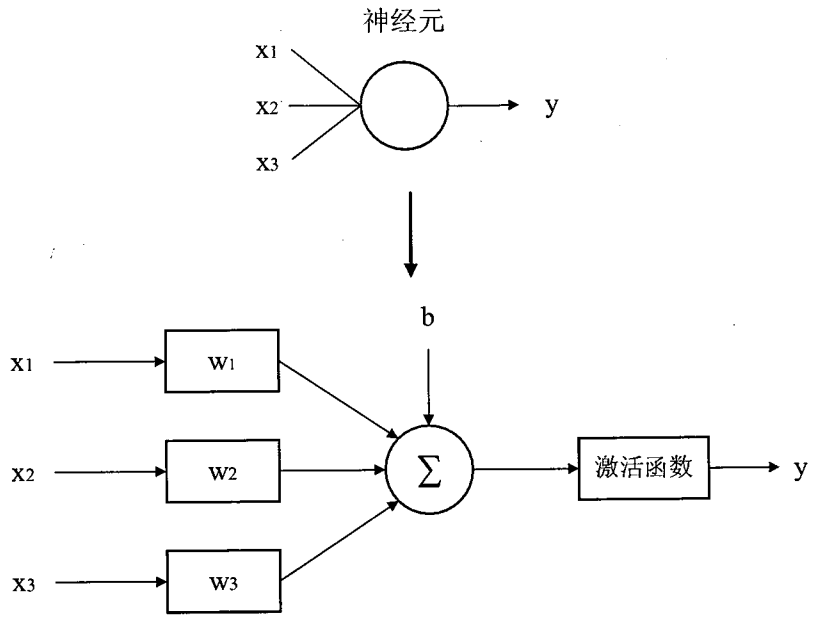


图 2.7　神经元参考模型

Figure2.7　Neuron Reference Model

在神经元参考模型中，整个输入输出的数学关系可通过公式（2.1）和（2.2）来进行表示：

$$U = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b \quad \text{...(2.1)}$$

$$y = f(U) \quad \text{...(2.2)}$$

其中，f 一般会选用 Sigmoid[32]、Tanh[33]和 ReLU[34]等。Sigmoid 函数的输出范围在 0 和 1 之间，其中，较大的输入会被映射到数值 1 附近，较小的输入则会向 0 趋近，适应于二分类问题，但当输入值过小或者过大时，函数的梯度会变得比较小，产生公式(2.3)所示的软饱和性现象，Sigmoid 函数对应的公式为(2.4)；Tanh 的输出以零为中心，关于原点中心对称，解决了 Sigmoid 非零中心的不足，但同样具有软饱和性，对应的公式如（2.5）所示，可通过 Sigmoid 进行表示；ReLU 将正的输出映射为输入本身，其它输入均输出 0，解决了 Tanh 和 Sigmoid 中经常发生的梯度消失问题，计算效率较高，可使网络模型更快的收敛，广泛被采用，公式如（2.6）所示；在 ReLU 之后，又出现了 LReLU、PReLU、ReLU6、Swish 和 Mish 等适应不同情况的激活函数。

$$\lim_{x\to\infty} f'(x)=0 \qquad ...(2.3)$$

$$\sigma(x)=\frac{1}{1+e^{-x}} \qquad ...(2.4)$$

$$tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=2sigmoid(2x)-1 \qquad ...(2.5)$$

$$f(x)=\max(0,x) \qquad ...(2.6)$$

2.2.2　卷积神经网络

在图 2.6 所示的神经网络结构中，采用的是全连接的连接方式，该种连接方式在层内神经元数量和层数不断增长的情况下，会产生大量的参数。例如，以 300×300 的数字图像作为图2.6网络结构的输入，并将单个像素看作一个输入结点，那么输入层就会产生 9 万个结点，此时的第一个隐含层如果只含有 2000 个结点，那么两者之间将包含 1 亿 8 千万个权重变量。

视觉生理学相关研究认为，人脑对外界事物的认知是从局部到全局，有关视觉感知的神经元也只对部分特定区域的响应产生刺激，具备局部感受野。卷积神经网络通过将网络结构中的全连接变为部分连接，只感知输入数据的局部特征，并在网络的深层次部分，对获得的特征进行综合，从而得到数据的整体信息，不仅符合人脑对视觉信息的认识方式，还大大降低了网络的权重数量。因此，在图像分类、图像语义分割和视频目标跟踪等计算机视觉相关的领域中，卷积神经网络结构的运算要更加高效。

该网络具有代表性的结构如图 2.8 所示。在对一副人像图片进行处理的过程中，浅层的网络可以获得低级的边缘信息，中间层能够得到相对精细的特征信息，例如面部的眼睛、鼻子和眉毛等五官，在后续更深层的网络中，可以获得类似人脸的高层次语义特征，如图 2.8 中右上角所示的特征图。在该特征图中，可以看到很多包含五官，较为完整的脸部特征信息。这些深层次特征信息，在该网络结构中，已经具备了初步的人脸辨识能力。

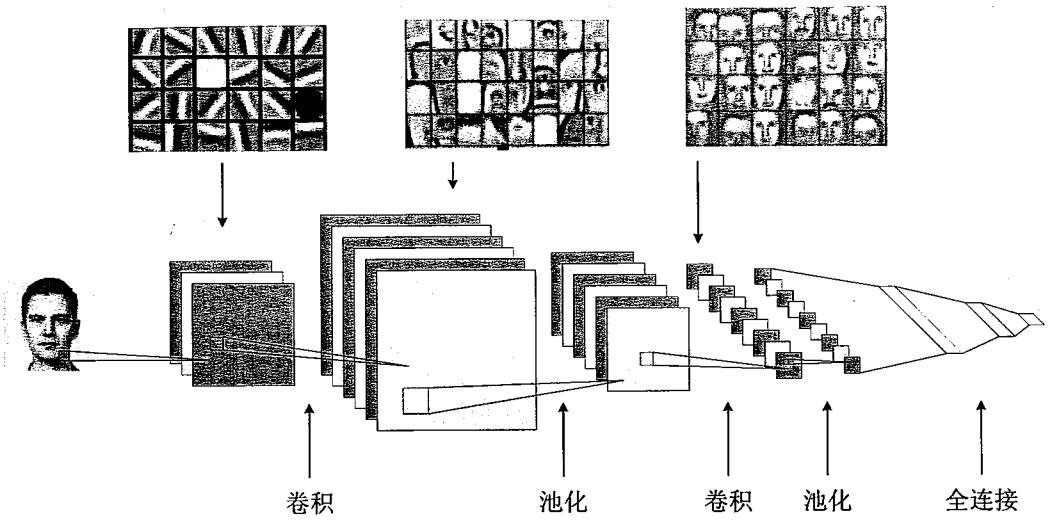


图 2.8　卷积神经网络的典型结构

Figure2.8　Typical structure of convolutional neural network

（1）卷积层

卷积层作为卷积神经网络的核心组件之一，通过其包含的卷积核对接收到的数据执行卷积操作，并将处理后的信息传递给下一层，整个流程类似人脑中视觉神经元对特定区域的响应 [35]。卷积层中的卷积核，尺寸一般相对较小，具有比较小的感受野，但会扩展到输入数据的整个通道内，即增加卷积核的深度。单个卷积核通过在二维数据的宽度和高度上移动，与输入数据进行卷积，得到对应的特征图，公式如（2.7）所示。I 代表二维输入数据，K 代表二维的卷积核，O 代表卷积后的特征输出。当把每个卷积核对应得到的特征图沿通道（深度）方向进行堆叠，就获得了卷积层完整的特征输出。

$$O(i,j)=(I*K)(i,j)=\sum_{m}\sum_{n}I(i+m,j+n)K(m,n) \qquad ...(2.7)$$

以二维的平面数据为例，大致的运算流程如图 2.9 所示。其中，单个的卷积输出是通过卷积核与其范围覆盖内的输入数据进行对应乘积求和得到的，如图 2.9（a）所示。如果卷积核每次以一个数据单位不断进行滑动，对输入的数据分别执行卷积操作，就可以得到最终的卷积输出，输出结果如图 2.9（b）。

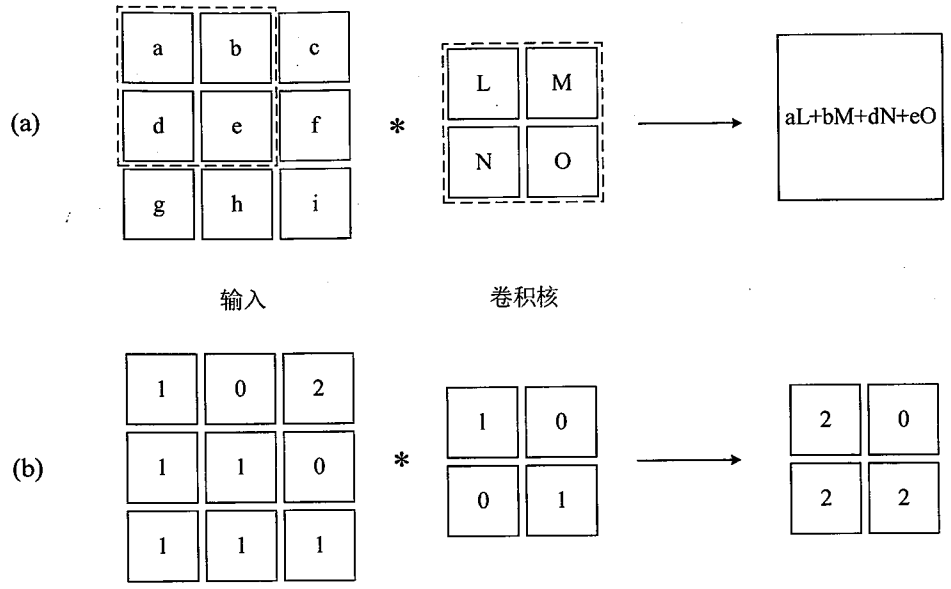


图 2.9　卷积运算示意图

Figure2.9　Schematic diagram of convolution operation

注：(a).单个输出结果情况下，虚线框内输入数据与卷积核的卷积原理 (b).输入数据与卷积核的完整卷积输出

Note：(a).In the case of a single output result, the convolution principe of the input data and the convolution kernel in the dotted line (b).The input data and the complete convolution output of the convolution kernel.

（2）池化层

卷积神经网络模型中另一个重要的构成为池化层。池化层通过下采样的操作，有效的减少了网络结构中的参数量，增强了网络对输入数据平移、形变的鲁棒性 [36]，有利于获得不因数据尺寸信息而发生变化的等效数据表征。池化层在实际的使用过程中，不包含通过训练习得的参数，仅需指定池化操作的类型、移动步幅和池大小就可以正常运行。常用的池化类型如图 2.10 所示。其中 2.10（a）展示的是池大小 2×2，步幅为 2 的最大池化；2.10（b）表示的则是池大小为 2×2，步幅为 2 的均值池化。

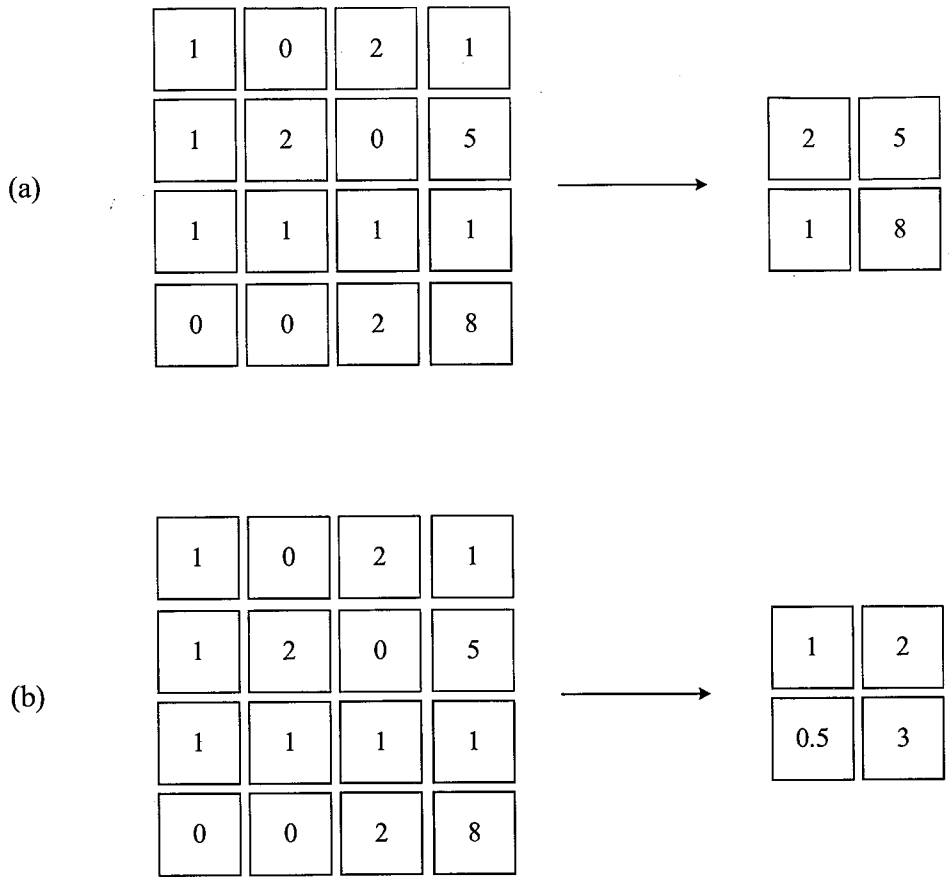


图 2.10　池化操作示意图

Figure2.10　Diagram of pooling operation

注：(a).最大池化操作 (b).平均池化操作

Note： (a).Maximum pooling operation (b).Average pooling operation

2.3　本章小结

本章首先对基于国产 AI 芯片的智能处理平台进行了简介，包括核心芯片的基本参数和主要功能，平台内部整体的媒体控制流程和处理方式，以及核心的智能视觉异构加速平台。其中，重点讲解了加速平台内部的 NNIE 硬件单元，阐述了该单元在 Caffe 模型下的具体开发模式和流程。之后，对神经网络的相关理论知识进行了介绍，对比了应用较为广泛的卷积神经网络和一般网络的区别，并对卷积网络中的核心构件进行了简单介绍。

第 3 章　基于海思芯片的智能处理平台搭建

3.1　智能平台总体设计

智能处理平台主要用于图像的目标检测。为了在嵌入式边缘设备上实现图像的采集、处理和显示，设计了如图 3.1 所示的处理流程。图像采集部分通过摄像头实时获取场景信息，并将得到的数据经由 MIPI（Mobile Industry Processor Interface）接口传送给平台核心的计算处理部分，即 Hi3559AV100 芯片；芯片接收到数据后，通过内部专用的神经网络处理单元，进行推理检测，并将检测结果通过 HDMI 接口在显示器上进行显示。作为整个平台的核心部分，Hi3559AV100 芯片在对输入场景进行检测的同时，还实现了整个平台的数据传递、系统调度和通信等功能。

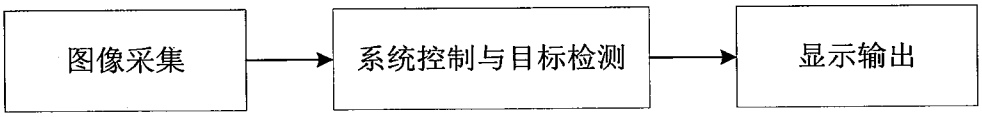


图 3.1　平台总体处理流程

Figure3.1　The overall processing flow of the platform

3.2　智能平台硬件部署

根据智能平台的设计流程，硬件单元应具备高集成度、低成本和小型化的特点。因此，本文选用如图 3.2 所示的硬件开发平台。编号“1”的部分是千兆以太网模块，可通过 TCP/IP 协议与外界设备进行通信，实现外部设备与该硬件平台内部系统的文件交换和交互操作。编号“2”是 Hi3559AV100 芯片，作为整个系统的核心，内部可实现海思媒体处理平台中的 VI、VPSS 和 VO 等功能模块，包含有 SVP 平台中的 DSP 和 NNIE 等智能加速引擎；在 NNIE 推理神经网络算法的过程中，可配置实现推理结果在 HDMI 接口（编号“5”）的直接输出；此外，通过其内部运行的嵌入式系统，可实现智能平台其它模块单元的控制。编号“3”的部分作为整个平台的外部存储器接口和板载存储单元，主要由一个 PCIe、两个 SDIO 3.0、DDR 和 eMMC 组成。

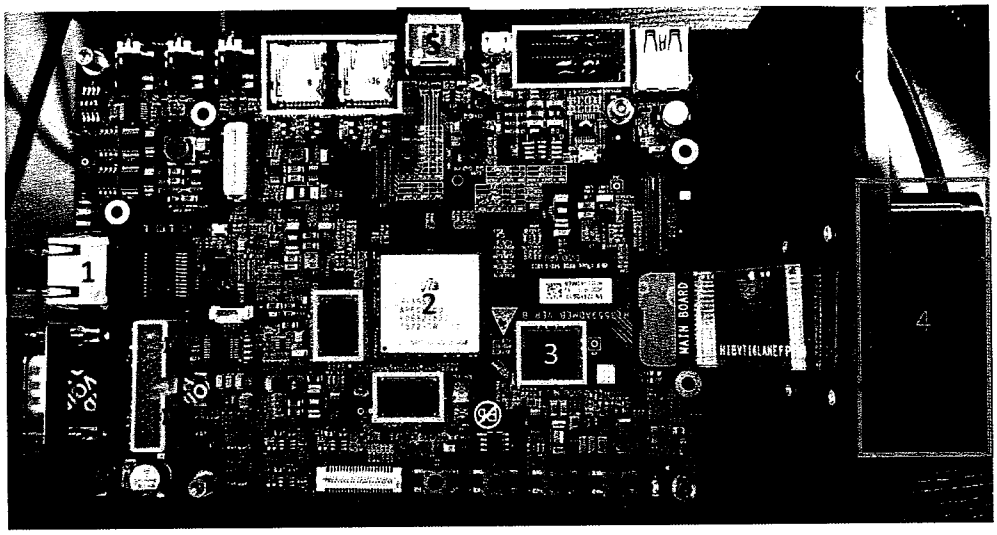


图 3.2　硬件开发平台

Figure3.2　Hardware development platform

编号“4”是整个平台的图像获取部分，核心为图像传感器。目前，主流的传感器主要分为CCD和CMOS两种，其中，CCD具有较高的动态范围，光线敏感度和信噪比较为优秀，而CMOS集成度高、体积小，并具备较低的功耗和应用成本，在技术不断的发展过程中，成像质量逐步提高[37][38]；经过综合比较，本文选用了索尼公司CMOS类型的IMX334图像传感器，该传感器的主要参数如表3.1所示[39]。

表 3.1　IMX334 图像传感器的主要参数

Table3.1　The main parameters of the IMX334 image sensor

参数名称	具备参数
传感器尺寸	8.86 mm（1/8 英寸）
单位像素尺寸	2.0 μm (H) x 2.0 μm (V)
总像素	3952(H) x 2320(V)
有效像素	3864(H) x 2180(V)
输入频率	6 ~ 27 MHz / 37.125 MHz / 74.25 MHz
支持 I/O	CSI-2 串行数据输出 （4 Lane / 8Lane, RAW 10 / RAW 12 输出）
A/D 转换	10 bit / 12 bit

工作温度范围	-10℃~ 60℃
高动态范围	函数多重曝光 HDR、数字重叠 HDR

3.3　开发环境集成

由于本文选用的海思平台资源有限，无法直接进行开发，因此，需要采用图 3.3 所示的开发模式[40]。Linux 服务器、Windows 工作机和目标平台彼此通过以太网实现互联，在 Windows 工作机和目标平台之间，还需增加串口的连通方式，以实现目标平台裸板情况下的程序烧写，以及 Windows 和目标平台之间的基础通信[41]。

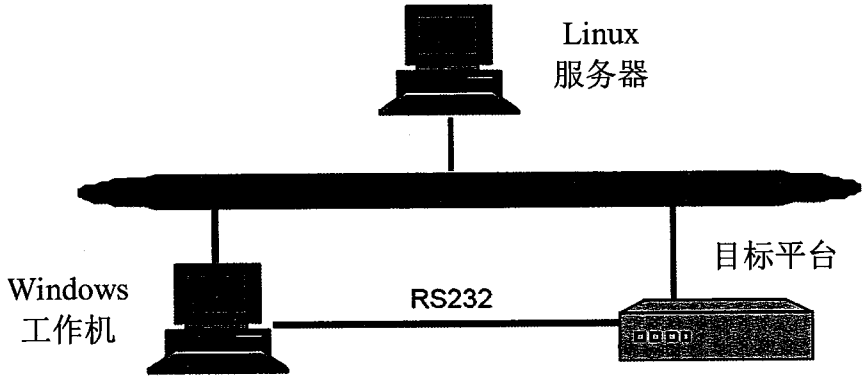


图 3.3　海思平台开发模式

Figure3.3　HiSilicon platform development model

3.3.1　交叉编译环境配置

在某个平台上编译生成可以在另外不同平台上运行的可执行文件的过程，称为交叉编译。交叉编译主要应用在嵌入式领域中，由于嵌入式目标平台资源有限，无法安装和运行针对自身平台的编译器和调试工具，因此，必须借助资源丰富的宿主机来实现编译和调试[42]。在本文的开发环境中，Linux 服务器充当宿主机的角色，其所需的交叉编译环境可通过图 3.4 所示的流程进行配置，具体配置步骤如下[40]：

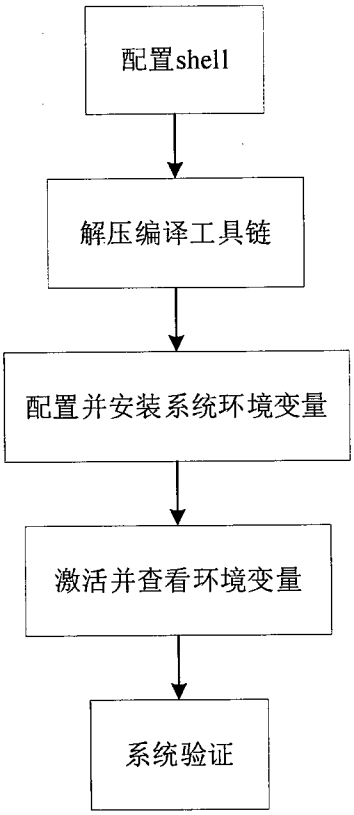


图 3.4 编译环境配置流程图

Figure3.4 The flow chart of compilation environment configuration

1、配置系统 shell。在 Linux 宿主机系统安装完成后，在 root 权限下输入 dpkg-reconfigure dash 命令，在弹出的提示框中选择 NO，即可配置默认 shell 为 dash；

2、解压工具链。在发布包中找到编译工具链对应的压缩包 aarch64-himix100-linux.tgz 和 gcc-arm-none-eabi-4_9-2015q3.tgz，对其分别进行解压，其中针对 64 位 Linux 操作系统的工具链为 aarch64-himix100-linux，gcc-arm-none-eabi-4_9-2015q3 为 Huawei LiteOS M7 的工具链，本文主要使用前者进行开发；

3、配置环境变量。将上一步解压得到的工具链，放置在一个固定的路径，并将该文件路径添加到系统的环境变量中。首先，在宿主机的/opt 目录下，新建 hisi-linux/x86-arm 的子目录；之后，将上一步解压得到的压缩包通过 tar –xf aarch64-himix100-linux.tar.bz2 –C /opt/hisi-linux/x86-arm 命令，解压压缩包中的内容到/opt/hisi-linux/x86-arm 目录下，同理，输入 tar –xzf runtime_lib.tgz –C /opt/hisi-linux/x86-arm/aarch64-himix100-linux，将 runtime_lib.tgz 压缩包内的内容解压在

指定的文件路径下；最后，通过 vim 编辑/etc/profile 文件，在文件的最后添加 export PATH=" /opt/hisi-linux/x86-arm/aarch64-himix100-linux/bin:$PATH"。

4、激活并查看环境变量。该过程主要由两个步骤组成，第一步，打开终端，键入 source /etc/profile，激活系统环境变量；第二步，验证环境变量在系统全局中是否可用，打开新的终端窗口，输入 echo $PATH | grep /opt/hisi-linux/x86-arm/aarch64-himix100-linux/bin，有输出则证明环境变量正确添加并激活。

5、系统最终验证，查看工具链是否可用。通过 aarch64-himix100-linux-gcc --version 命令，可以查看 gcc 版本，或者输入 aarch64-himix100-linux-按两下 Tab 键，终端能够输出一系列工具链的名称，则证明搭建完成。

3.3.2　NNIE 开发环境配置

在海思提供的异构加速平台 SVP 中，对智能加速硬件单元的开发和使用，需要对应的软件环境。在本文使用的 NNIE 硬件单元中，主要是配置工具链中的 mapper 工具。在 Windows 工作机上，通过安装 SVP 开发包中的 RuyiStudio 软件，即可实现所需环境的配置。RuyiStudio 软件内部集成了适配 Windows 操作系统的 mapper 工具和仿真库，能够实现 Caffe 算法模型到 wk 数据指令文件的转换和指令文件的仿真；除此之外，该工具还具备显示 Caffe 模型拓扑结构和对比模型输出向量相似度等能力。

该工具的整个安装过程分为编译链和依赖配置两个部分。编译链的安装主要分为以下过程 [31]：

1、进入 MinGW 的官网，选择 7.3.0 版本下的 x86_64-posix-she，下载并解压到 SVP 开发包中的 ruyi_env_setup 目录下；

2、下载 MinGW 的扩展文件 msys+7za+wget+svn+git+mercurial+cvs-rev13，并将其解压到 MinGW 的文件夹下；该扩展主要用于清理 RuyiStudio 工程缓存过程中用到的 rm.exe 文件；

3、在 Windows 的环境变量 Path 中，添加 MinGW 和扩展 msys 的 bin 路径，并在 MinGW 的 bin 目录下，将 x86_64-w64-mingw32-gcc.exe 文件拷贝重命名为 mingw32-gcc.exe。

在 MinGW-64 安装完成后，通过以下步骤完成依赖的配置：

1、在 Windows 环境变量中配置添加 RUYI_PYTHON_PATH 和用于识别 caffe 的 PYTHONPATH 变量，并将前者添加在系统环境 Path 中；

2、根据 ruyi_env_setup 目录下的 requirements.txt 文件，在对应网站上下载所需版本的依赖包，并将得到的压缩包解压在当前目录的 python35 文件夹下，其中，opencv_python、protobuf 和 caffe 的压缩包则需要通过步骤 3 单独进行配置；

3、将步骤 2 得到的 opencv_python、protobuf 和 caffe 压缩包，在 ruyi_env_setup 目录下的 python35\Lib\site-packages 中解压缩。之后通过 Windows 的 cmd 命令进入当前的解压目录，使用 pip 指令对 opencv_python 和 protobuf 进行安装；在 caffe 解压完成后，将步骤 2 下载的 libraries 依赖包内的部分 dll 文件拷贝到 caffe 解压目录的 python\caffe 路径下；

4、安装 VS 软件；执行 ruyi_env_setup 目录下的 setup_roi_caffe 指令。VS 安装成功后，对指令下载得到的 py-faster-rcnn-windows-master 文件进行编译，并将编译后的文件拷贝到 caffe 的 roi_pooling 目录下。

至此，在 RuyiStudio 工具中进行 NNIE 开发所需的 MinGW-64 编译链和环境依赖就安装配置完成了，之后，重启计算机即可生效环境配置。

3.3.3 整体开发环境配置

在前文 Linux 服务器交叉编译环境配置成功的基础上，还需要对海思提供的开发包 SDK（Software Development Kit）进行安装。Hi3559AV100 开发包在 Linux 服务器上的具体安装步骤如下 [43]：

1、在软件包 01.software/board 目录下，找到 Hi3559AV100_SDK_V2.0.3.0.tgz，并将其拷贝在 Ubuntu 的 Linux 开发机上；

2、运行 tar –zxf Hi3559AV100_SDK_V2.0.3.0.tgz 命令，解压压缩包，等待解压过程完成；

3、进入解压后的 Hi3559AV100_SDK_V2.0.3.0 目录，使用 root 权限运行./sdk.unpack 命令，展开 SDK 包内压缩存放的内容。

在软件包安装完成之后，为了实现高效和稳定开发，还需要针对 Linux 服务器、Windows 工作机和单板平台配置对应的软件依赖。三者各自的软件环境配置如下：

Linux 服务器：使用 Ubuntu 16.04 操作系统，配置有 vim、arm 交叉编译环境（gcc 6.3.0）、telnet 服务、NFS、samba 等；

Windows 工作机：安装 Windows 10 操作系统，需要 SecureCRT、HiTool、RuyiStudio、Xshell、Source Insight、Visual Studio 等软件；

Hi3559AV100 平台：Uboot 作为引导程序，基于 4.9.y 版本 Linux kernel 移植开发的 Hisilicon Linux 操作系统；由 busybox 1.26.2 制作得到的文件系统，包含常用的 Linux 命令。

3.4　系统编译和移植

在前述开发环境搭建完成的基础上，为了使目标平台高效地运转，需要在平台的裸板上烧写嵌入式 Linux 操作系统 [44]，该系统在存储介质中的分布结构如图 3.5 所示。

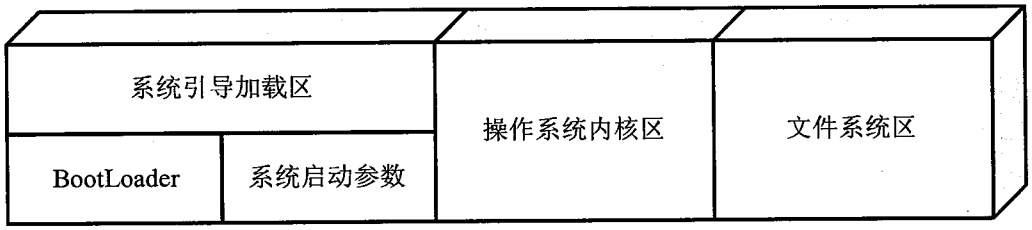


图 3.5　嵌入式 Linux 系统的分区结构

Figure3.5　The partition structure of embedded Linux system

系统引导加载区主要存放 BootLoader 与启动参数，针对目标嵌入式平台的 Linux 内核和根文件系统则分别存储在操作系统内核区和文件系统区 [45]。接下来，针对海思平台，对三个分区分别进行配置和烧录。

3.4.1　BootLoader 选取与烧写

系统引导加载区为平台上电后，进行的第一步操作，其中的 BootLoader 作为调用操作系统之前的一个启动程序，为操作系统内核的上载运行做前序的配置工作，实现了软件系统与硬件单元的初次连接，对后续软件平台的正常运转和开发起到了重要的铺垫作用 [46]。BootLoader 在运行过程中，完成的主要操作囊括了以下几点，首先，初始化目标板的硬件设备，包括看门狗和中断的关闭、系统工作时钟的配置和存储介质控制单元的初始化等，其次，为内存空间建立映射图，

实现多个程序之间有效安全的共享存储，最后，构建一个合适的系统软硬件环境，为操作系统内核的加载运行做准备[47]。

BootLoader 一般可分为两种不同的工作模式。在第一种启动加载模式下，整个启动过程不需要用户的干预，在板子上电后，存储在存储介质中的操作系统将通过 BootLoader 加载到目标平台的内存上运行，在产品开发完成，进行发布时，选用该工作模式；第二种的下载模式，主要应用在开发人员的开发和测试过程中，通过使用相应的命令，将目标平台与宿主机以串口或网络等通信方式进行连接，并从宿主机传输操作系统内核、根文件系统等文件到目标平台的内存中，然后，根据实际需要，在内存中运行或者烧写在其它存储介质中。目前，BootLoader 的种类繁多，各具特色，在具体的使用范围和架构支持上略有差别[48]，表 3.2 列出了开放源码的部分引导程序。

表 3.2 开放源码的引导程序[49][50]

Table3.2 Open source bootloader

类别	描述	ARM 平台	X86 平台	PowerPC
LILO	Linux 磁盘的引导程序	否	是	否
GRUB	LILO 在 GNU 下的替代程序	否	是	否
LinuxBIOS	替代 BUIS 的 Linux 引导程序	否	是	否
ROLO	省去 BIOS，直接从 ROM 引导 Linux	否	是	否
BLOB	LART 等其它硬件平台的引导程序	是	否	否
Vivi	针对三星 ARM 处理器的引导程序	是	否	否
RedBoot	基于 eCos 的引导程序	是	是	是
U-Boot	通用引导程序	是	是	是

在本文使用的海思平台上，选用的 BootLoader 为 U-boot，它是遵循 GPL 的开源项目，前身是德国创建的 PPCBOOT 工程，经过了不断地发展，在实际应用中稳定可靠，适应性强，并具有丰富的开发调试文档和设备驱动源码（包括串口、以太网和 RTC 等），适应系统不同开发阶段的调试和发布需求。在 U-boot 烧录进启动介质之前，需要针对目标平台对其进行移植[51]。

本文针对海思平台的移植过程，主要包括编译 U-boot、配置 DDR 存储器和硬件的管脚复用（可选）。其中，在 U-boot 编译的过程中，选用了 eMMC 作为系统的启动介质；因此，需要通过 make CROSS_COMPILE=aarch64-himix100-linux- hi3559av100_emmc_defconfig 命令，在 Linux 服务器上配置 eMMC 所需的编译环境；在环境配置成功后，运行 make CROSS_COMPILE=aarch64-himix100-linux- -j 10 命令，编译 U-boot；此时，在文件夹下会得到对应的 u-boot.bin，然而当前获得的 bin 文件并不能充当目标平台最终烧录的镜像，只是作为一个中间件而存在。在这之后，需要进入开发包下的 uboot_tools 文件夹，打开对应的 U-boot 配置表格，在 Windows 工作机下，对表中与 DDR 相关的内容进行修改，从而匹配单板所选用的 DDR 存储器；此时，如果还需修改管脚复用，对管脚复用的配置表格进行更改即可；配置完成之后，对表格进行保存，并在表格第一个标签页上点击 Generate reg bin file 按钮，在当前文件夹下生成 reg_info.bin 文件，随后，将该文件重命名为.reg，并拷贝到开发包 osdrv/opensource/uboot/u-boot-2016.11 的目录下，执行 make CROSS_COMPILE=aarch64-himix100-linux- u-boot-z.bin 命令，即可在当前文件夹下生成最终所需的 U-boot 镜像 [52][53]。

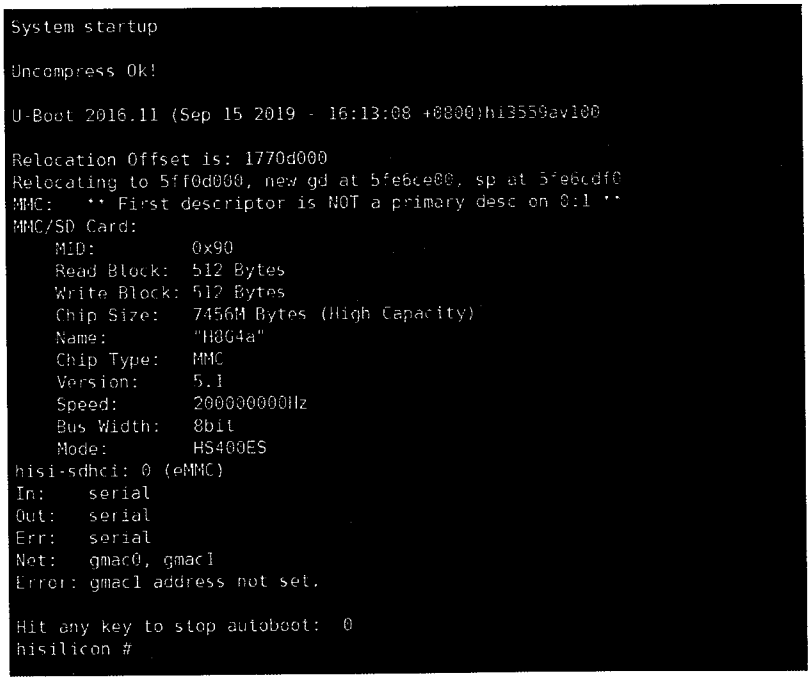


图 3.6　U-boot 成功烧录后的启动界面

Figure3.6　The boot interface after U-boot is successfully burned

获得镜像文件后，第一次的烧录过程，需要使用开发包中提供的 HiTool 工具来完成。在 Windows 工作机上打开 HiTool 工具。首先，选择目标平台的芯片型号，并在 PC 与板端配置界面中，配置单板连接所用的串口、MAC 地址、IP 地址和网关等；随后，根据所选用的 eMMC，切换到 Burn eMMC 模式下，载入上一步移植成功的 U-boot 镜像文件，并配置所需的分区信息；最后，在硬件单元下电的情况下，单击烧录按钮并重新上电，等候 U-boot 镜像文件烧录完成。之后，重启单板，得到图 3.6 所示的界面，即证明移植后的 U-boot 程序成功烧写在了目标平台上[54]。

3.4.2 Linux 内核配置与编译

在操作系统领域，Linux 凭借其源代码开源、定制和裁剪便捷、适应性强、稳定可靠和成熟配套的开发工具，在嵌入式领域得到了广泛的应用，例如路由器、机顶盒、智能电视、数码钢琴和舞台灯光控制系统等[55]。嵌入式 Linux 内核的核心源程序按照树形结构的方式进行组织[56]，与 PC 平台上的 Linux 操作系统相比，两者的程序启动流程和原理大体相同[57]。

在上文 U-boot 移植与烧录完成后，本文配置和编译了 4.9.37 版本的 Linux 内核。首先，从 Linux 开源社区下载对应版本的内核源码，在平台开发包提供的内核补丁下，通过 patch 命令实现内核源码打补丁的操作；接下来，在补丁完成后的源代码目录下，手动拷贝启动介质为 eMMC 的.config 配置文件，并执行 make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- menuconfig 命令[40][58]，弹出如图 3.7 所示的配置界面，该界面是基于文本的图形化配置菜单，在 Linux kernel 的配置和裁剪过程中被广泛使用。界面菜单中的默认选项能够满足大多数平台的基本需求，在实际地配置过程中，可通过撤销部分不需要的功能选项或者将使用频率较低、与其它功能关系较弱的功能编译成动态加载的模块单元，实现内核的裁剪，从而达到减小内核体积和内存消耗的目的[59][60]。

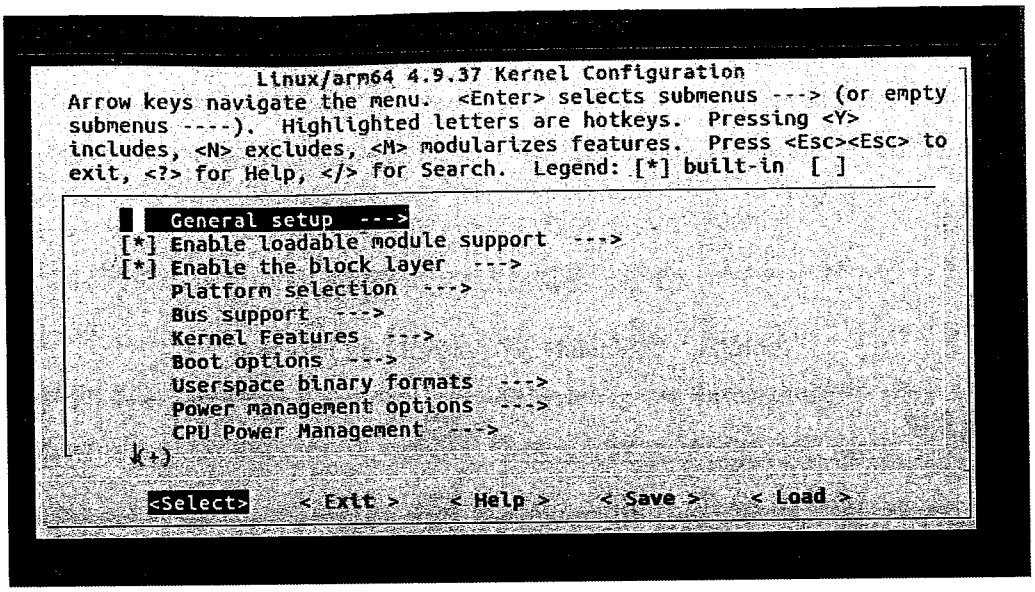


图 3.7　Linux 内核配置的图形化界面

Figure3.7　Graphical interface for Linux kernel configuration

内核配置完成后，通过新生成的.config 文本，对旧的配置文件进行覆盖；随后，在开发包的 osdrv 顶层目录下，执行 make BOOT_MEDIA=emmc AMP_TYPE=linux atf 命令，或者在终端的命令窗口输入 make ARCH=arm64 CROSS_COMPILE=aarch64-himix100-linux- uImag 指令，生成初步的 uImage 镜像文件；接下来，需要在开发包的 osdrv/opensource/arm-trusted-firmware/arm-trusted-firmware 目录下，配置并执行 mk.sh 脚本，在该脚本成功运行后，即可得到目标平台最终所需的 uImage 镜像文件 [40]。

3.4.3　根文件系统选择与编译

在计算机中，数据的检索和存储由文件系统进行控制。文件系统通过把数据分隔成不同的部分并给不同的部分命名形成树形目录，实现了数据的分离和识别，否则，计算机中的数据将被放入一整块存储介质，无法在使用过程中有效的辨别数据的大小和存取地址；其中，数据被划分的不同部分被称为文件，而管理文件及其目录的逻辑规则和结构，被称为文件系统 [61][62]。在文件系统形成的树形逻辑结构中，根目录处于最顶端，在其对应的物理介质中，存储着根文件系统。

根文件系统存放有操作系统正常运行所必须的文件，包括所有的系统命令、系统配置、系统运行所需要的链接库和应用程序等。在嵌入式领域，为了缩减文

件系统的大小，精简根文件系统，通常会忽略一部分文件目录[63]，例如，用于给多用户提供扩展的/opt、/mnt和/root，以及存储用户某些临时文件的/tmp目录等[40]。目前，在嵌入式领域，具有不同种类的文件系统，每种文件系统具有不同的特性，比较常用的文件系统如表 3.3 所示。

表 3.3 常用嵌入式文件系统

Table3.3 Commonly used embedded file system

文件系统	描述
cramfs	针对Linux 2.4版本内核之后的文件系统，采用压缩的形式存储文件数据，节约存储空间，但只可读取且读取效率低。
jffs2	基于jffs[64]的改进版本，是日志结构化的读写文件系统；会造成Flash空间的浪费，并在分区较大的情况下，消耗较长的挂载时间。
initrd[65]	相当于存储介质，可支持ext2[20]、cramfs等格式；需要内核支持多嵌入式文件系统，在内核的缓存机制下，会浪费一定的存储空间。
yaffs2[66]	针对NAND Flash存储介质，属于日志结构的文件系统；运行和启动速度快，具备掉电保护的功能，但未采用压缩文件格式，镜像文件较大，只适用于NAND Flash。
squashfs[67]	属于Linux内核适用的只读形式的压缩文件系统；压缩率高，可实现重复文件数据的检测和删除。
ext4[20]	ext2和ext3的改进版本，具备延迟获取存储空间的功能，可增加存储介质的性能，减少文件的分散度；能够支持大容量的分区，可向下兼容ext2、ext3，适用于eMMC和UFS。

结合表 3.3 和本文平台采用的存储介质，最终选用 ext4 作为目标平台的文件系统。ext4 文件系统的制作流程如图 3.8 所示。

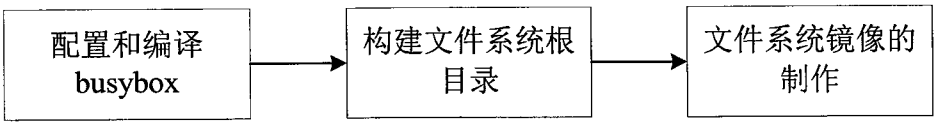


图 3.8 ext4 镜像的制作流程

Figure3.8 ext4 mirror production process

首先，需要对 BusyBox[68]工具进行配置。在开发包的 osdrv/opensource 目录下，找到 BusyBox 的源代码，通过 cp 命令拷贝对应的工具链配置文件，随后，在终端中运行 make menuconfig 命令，得到图 3.9 所示的图形化配置界面。

在 Busybox Setting 选项下的 Build Options 中，选择推荐的交叉编译器，其余功能根据所用平台的需求进行配置；在配置完成保存退出后，通过终端输入 make 和 make install，分别实现 BusyBox 的编译和安装；随后，将在当前目录下获得初级根文件系统，内部仅包含/bin、/sbin、/usr 和 linuxrc[69]，接下来，拷贝其内部的全部内容到新的 rootfs 文件夹下，并在当前位置使用 mkdir 命令新增必要的目录，目录创建完成后，对添加的/etc、/lib、/dev 等目录进行配置，其中，/etc 目录下主要包括 fstab、inittab、init.d/rcS 等文件，/lib 内则用于保存应用程序运行的依赖库 [57]；目录配置完成后，就实现了根文件系统的构建。接下来，利用该系统制作 ext4 镜像；在开发包的 osdrv/tools/pc/ext4_utils 目录下，通过配置执行 Makefile，生成制作工具 make_ext4fs，得到制作工具后，通过./make_ext4fs -l 96M -s osdrv/pub/rootfs_hi3559av100_96M.ext4 osdrv/ rootfs 命令，便可得到最终的 ext4 文件系统镜像。

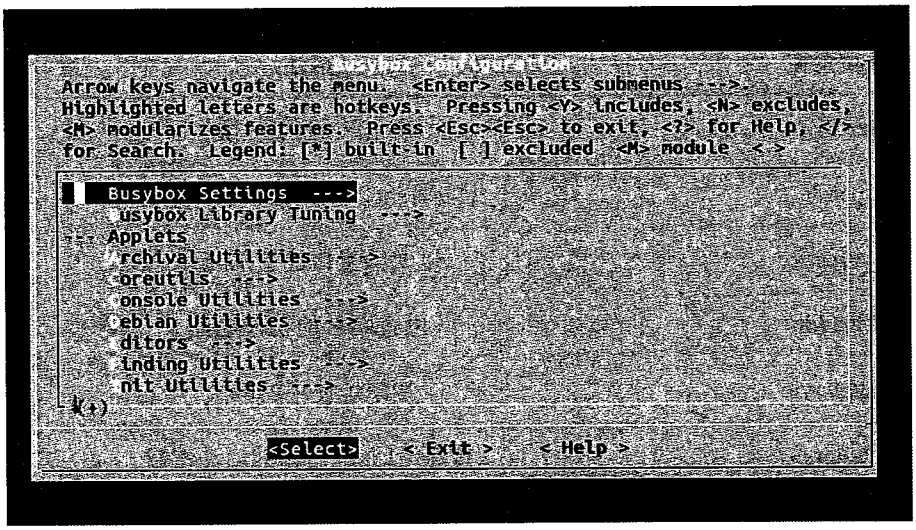


图 3.9　BusyBox 图形化配置界面

Figure3.9　BusyBox graphical configuration interface

3.5 内核和根文件系统的烧写

经过本章前述内容的配置和编译，目前已完成 U-boot 程序的烧录，并得到了 Linux 内核和 ext4 的镜像文件；接下来，需要在目标平台上完成后面两者的烧写。

整个过程可通过 U-boot 程序或 HiTool 工具进行实现。在 U-boot 程序的实现方式中，通过海思平台开发模式中的串口登录到目标平台，在 U-boot 的命令窗口下，通过命令行的方式完成环境变量的配置和镜像文件的烧写；在 HiTool 工具中，则需要按照嵌入式系统的实际需求，在其配置界面进行相应的参数配置，完成烧写。但是，在 HiTool 工具的运行过程中，文件烧录的本质也是通过传递 U-boot 命令来完成的。因此，本文选用第一种方法来实现镜像文件的烧录。

首先，需要使用 setenv 命令，对目标板和网络接口有关的环境变量进行配置，实现 Windows 工作机和目标板的网络互联和文件传输。其中，U-boot 程序常用环境变量如表 3.4 所示。

表 3.4 U-boot 常用环境变量 [70]

Table3.4 U-boot commonly used environment variables

环境变量	描述
Serverip	宿主PC机的IP地址
Ipaddr	目标平台以太网接口的IP地址
Netmask	以太网接口的子网掩码
Ethaddr	以太网卡的物理地址
Gatewayip	IP地址使用的网关
Bootargs	传递给Linux内核的启动参数
Bootdelay	配置自启前需要等待的秒数
Bootcmd	等待够bootdelay延时时间后，自动执行的命令
Bootm	从内存的指定位置处运行镜像文件

在通过 ping 测试网络链接成功后，通过内存写入命令 mw.b、文件传输协议 tftp 以及 MMC 子系统中的 write 和 write.ext4，在 eMMC 上实现 Linux kernel、ext4 文件镜像的烧录；最后，通过命令 setenv bootargs 'mem=512M

console=ttyAMA0,115200 root=/dev/mmcblk0p3 rootwait rw rootfstype=ext4 blkdevparts=mmcblk0:1M(boot),10M(kernel),96M(rootfs)'、setenv bootcmd 'mmc read 0 0x44000000 0x800 0x42ab;bootm 0x44000000'和 saveenv，对内核的启动参数与命令进行配置和保存。

3.6　本章小结

本章内容主要围绕智能处理平台的搭建工作进行展开。首先，对平台的总体处理流程进行了设计，并根据该流程确定了具体的硬件单元；紧接着，对智能处理平台所需的集成开发环境进行了配置和搭建，并在该环境基础上，制作了适配平台的 BootLoader、Linux 内核和 ext4 镜像；最后，通过 U-boot 指令，将上述文件在目标平台上进行了烧写。

第 4 章　基于海思智能处理平台的目标检测

4.1　目标检测算法

4.1.1　传统目标检测算法

在计算机视觉相关技术的不断发展过程中，目标检测经历了传统和基于深度学习的两个主要历史发展时期。传统的算法主要分成了候选框生成、特征向量提取和分类三个部分，如图 4.1 所示。

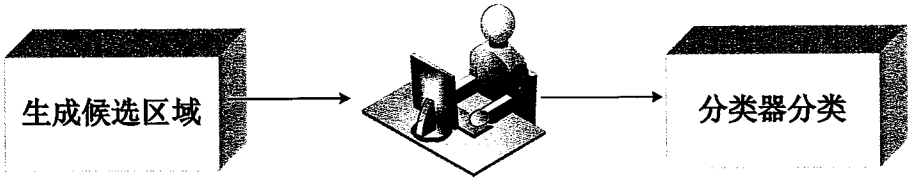


图 4.1　传统目标检测算法流程

Figure4.1　The flow of traditional object detection algorithm

在候选框生成阶段，通过滑动窗口搜索整个待检测图像，找到可能含有目标的感兴趣区域[71][72]，由于实际的目标物体在图像中可能存在不同的宽高和比例，通常会对输入图像的大小进行调整，并选用多尺度的滑窗进行搜索。在特征向量提取部分，将上一阶段获得的候选区域作为输入，对其进行特征的提取，以得到该区域的语义特征；这些特征通常由特征提取器进行提取，其中常见的提取算法有 SIFT[73]、HOG[74]、Haar[75]和 SURF[76]等。在得到了候选区域的特征后，通过训练后的分类器进行分类，从而实现最终的目标检测；在该过程中，常用的分类器有 SVM[77]、adaboost[78]和 bagging[79]等。但在传统目标检测算法检测的过程中，会产生以下一些问题：首先，在提取候选框的过程中，会产生大量冗余的候选框，导致检测错误，不仅如此，候选框的尺度选择属于启发式的，在某些情况下，无法很好的匹配目标物体，此外，特征提取器基于手工设计，在对应的特定场景下才能取得较好的检测结果，无法在场景复杂的情况下提取关键的特征信息。

4.1.2 基于深度学习的目标检测算法

基于深度学习的目标检测算法，主要由深度卷积神经网络组成，与传统检测算法中基于手工设计的特提取器相比，卷积神经网络可以通过训练，生成从原始像素信息到深层语义信息的多层特征表示，且能在复杂的场景下提取更加有效的特征信息。除此之外，相比传统算法在大量数据下，固定的学习能力，卷积神经网络可通过训练，习得更好的特征表示。

目前，基于深度学习的目标检测算法主要分为两阶段模型和单阶段模型两种。两阶段算法模型将目标检测任务分为候选框获取和区域预测两个主要过程。在候选框获取过程中，检测器将识别出输入图像中可能存在目标物体的区域，并产生高召回率的候选框；在区域预测阶段，使用基于深度学习的算法模型对候选区域进行分类，并对第一阶段生成的候选结果进行反馈微调，其中，被分类的候选框可能包含期望的目标物体，也可能仅仅囊括了图片背景，不管是目标类别还是背景信息，在该过程中都将被有效的检测。与两阶段算法相比，单阶段算法模型没有生成候选区域的独立过程，整个算法将输入图像上的所有位置都视为潜在的目标物体位置，并对每个感兴趣区域进行背景或目标物体类别的划分。在算法的推理过程中，两阶段模型通常在检测速度上会慢于单阶段模型，因为两阶段模型包括候选框提取和区域分类两部分，这使得它们相比于单阶段模型通过单一的网络，直接从图像得到检测结果，在计算上耗费更长的时间[36]。

4.1.3 目标检测数据集与评价指标

数据集在目标检测算法的发展过程中，起到了很好的推动作用，在大量数据的训练下，网络模型能够习得目标物体的一般化特征，获得较好的泛化特性。目前，知名的数据集主要包括 Pascal VOC[80]、MS COCO[81]和 ImageNet[82]等；表 4.1 对该类数据集进行了统计。

表 4.1 目标检测数据集的统计结果

Table4.1 The statistical results of object detection datasets

数据集	类别	Train 张数	Validation 张数	Test 张数
Pascal VOC 2007	20	2501	2510	4952
Pascal VOC 2012	20	5717	5823	10991

ILSVRC 2013	200	395909	20121	40152
ILSVRC 2017	200	456567	20121	65500
MS COCO	80	118287	5000	40670
Open Images	600	1743042	41620	125436

为了评测检测算法的性能，通常采用表 4.2 所示的评价指标对算法模型进行评估。表中的 FPS 和 BFLOPS 主要用于评估算法的检测速度和计算复杂度，剩余的指标则用来评价算法的检测精度。

表 4.2　目标检测算法常用的评价指标

Table4.2　Common evaluation metrics for object detection algorithms

指标	全称	定义和描述
TP	True Positive	正确分类的正样本
TN	True Negative	正确分类的负样本
FP	False Positive	错误分类的负样本
FN	False Negative	错误分类的正样本
AP	Average Precision	不同召回率下的平均检测精度，通常在特定类别下进行评估
mAP	mean AP	所有类别的平均 AP
FPS	Frame per second	每秒钟处理的图片数
BFLOPS	Billion Float Operations	十亿次浮点运算

4.2　算法选取

通过上一章内容，已将完成了海思智能处理平台软硬开发环境的搭建，以及操作系统的烧写和上载。接下来，在基于国产 AI 芯片的智能处理平台上，对基于深度神经网络的目标检测算法进行了部署研究。为了验证该智能处理平台的普适性，考虑选用深度学习领域比较流行的两阶段和单阶段目标检测算法模型进行部署实现。

4.2.1 两阶段算法模型

R-CNN[83]是最早的两阶段检测模型，它首先通过选择性搜索算法 [84]生成一组稀疏的候选框，在输入图像中将每个候选框所选中的区域裁剪下来，调整为相同的尺寸大小，然后，分别经过卷积网络和 SVM 对相同大小区域的特征进行提取和分类。R-CNN 在 Pascal VOC2010 中，相比之前最优的传统目标目标检测算法 SegDPM[85]，在检测结果方面有了较大的提升。但是，R-CNN 也存在一些问题。卷积神经网络分别获取单个候选区域特征的过程，产生了大量的重复计算；不仅如此，R-CNN 的几个检测阶段也相对独立，无法以端到端的方式进行优化，很难获得全局最优解；除此之外，选择性搜索算法也主要依赖低级的视觉信息，在复杂的场景下，无法获得有效的候选框。因此，在这之后，融合空间金字塔 [86]的 SPP-net[87]被提出。

相比 R-CNN，SPP-net 无需裁剪候选区并将其分别送入 CNN 进行特征提取，而是直接通过一个 CNN 结构提取整幅图片的特征，并通过空间金字塔池化层在特征图上获取特定维度的向量。此外，SPP-net 还可在不同纵横比的区域上进行处理，无需 R-CNN 中尺寸调整的操作，因此，SPP-net 不会遭受信息丢失和图像几何失真的情况。但是，SPP 算法模型依然由多个独立阶段组成，无法实现端到端的优化。在 SPP-net 之后，Fast R-CNN[88]通过固定比例的 ROI pooling 获取候选区内的特征信息，使用 softmax 函数实现最终的分类；在特征获取、区域类别划分和边框回归的过程中实现了端到端的优化，相比 SPP-net，检测精度和推理速度都得到了进一步的提升。但是，Fast R-CNN 的候选框依然是按照选择性搜索的原理得到的，存在 R-CNN 一样的不足。为了解决该问题，Faster R-CNN[89]提出了基于卷积结构的 RPN 网络，该网络通过$n\times n$的滑窗滑过目标图像的特征图，得到对应位置的特征向量。由于基于卷积结构，RPN 模型中的参数信息可通过监督学习的方法习得，除此之外，RPN 还可以和骨干网络共享卷积，使得整个检测模型能够做到端到端的优化。

在 Faster R-CNN 算法之后，两阶段算法模型经过不断地发展和创新，相继出现了 OHEM[90]、MR-CNN[91]、LocNet[92]和 R-FCN[17]等算法模型。目前，常见两阶段算法模型在 VOC 测试集上的对比结果如图 4.2 所示。

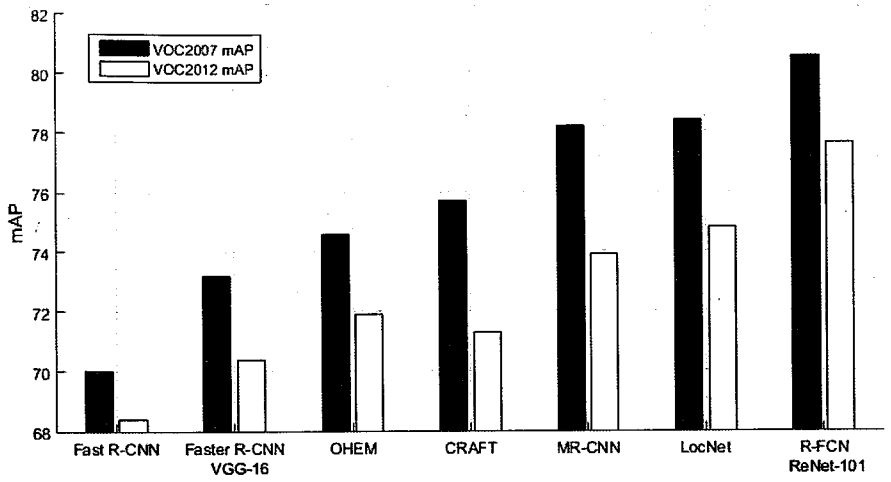


图 4.2　部分两阶段模型在 VOC 测试集上的对比结果

Figure4.2　Comparison results of some two-stage models on the VOC test set

上述算法模型在 GTX1080Ti 显卡下的详细检测结果如表 4.3 所示，其中包括了部分算法模型在不同骨干网络下的测试结果。其中，07+12 代表 VOC 2007 trainval 和 VOC 2012 trainval 训练集，对应的 mAP 为 VOC 2007 test 的测试结果；07++12 则表示 VOC 2007 trainval、VOC 2007 test 和 VOC 2012 trainval 的训练集，mAP 则对应 VOC 2012 test 下的测试结果。

表 4.3　部分两阶段模型在 VOC 测试集上的详细结果

Table4.3　Detailed results of some two-stage models on the VOC test set

Method	Backbone	Data(Train)	mAP(%)	Data(Train)	mAP(%)
Fast R-CNN	VGG-16	07+12	70.0	07++12	68.4
Faster R-CNN	VGG-16	07+12	73.2	07++12	70.4
Faster R-CNN	Residual-101	07+12	76.4	07++12	73.8
OHEM	VGG-16	07+12	74.6	07++12	71.9
CRAFT[93]	VGG-16	07+12	75.7	07++12	71.3
MR-CNN	VGG-16	07+12	78.2	07++12	73.9
LocNet	VGG-16	07+12	78.4	07++12	74.8
R-FCN	ResNet-50	07+12	78.7	07++12	75.1

R-FCN	ResNet-101	07+12	80.5	07++12	77.6

从图表中的结果可以看出，R-FCN 作为 Faster R-CNN 算法模型的改进版，相比其它两阶段算法，在 VOC 2007 test 和 2012 test 数据集上的检测结果提高最为明显；因此，本文考虑选用 R-FCN 网络模型在智能平台上进行部署。

4.2.2 单阶段算法模型

YOLO[94]作为较早的单阶段实时检测模型，通过省略候选框生成的步骤，将目标检测过程看作一个回归问题，在单个卷积网络下实现检测。YOLO 在空间上将输入图像划分为固定数目的网格，并把每个网格包含的图像空间看作候选区，在每个候选区内对目标对象进行检测，被检测的每个区域，最终会得到是否包含目标物体、物体类别以及边界框位置和尺寸的信息。但是，YOLO 也面临着一些问题，在每个被划分的网格中，最多只能检测两个物体，使得其在小目标和拥挤场景下的检测，变得比较困难，此外，整个检测过程仅使用单一的特征图，在多尺度目标的情况下，检测效果较差。

为了解决 YOLO 算法的不足，之后出现了新的单阶段算法模型 SSD[28]。SSD 同样使用网格将图片划分为不同的区域，但是，不同于 YOLO 在网格候选区域上直接进行预测，SSD 在每个网格中，通过一系列不同长宽和比例的锚点框（anchor box）进行预测；此外，SSD 在多个特征图上对目标物体进行检测，不同尺寸的特征图根据其具备的感受野，检测图片中特定比例范围的目标；最后，通过融合不同特征图的中间结果，得到最终的检测输出。表 4.4 展示了 GTX1080Ti 显卡下，应用比较广泛的 YOLOv3、STDN[95]、SSD 和其改进版 DSSD[96]在 MS COCO 标准数据集上的检测结果。

表 4.4 部分单阶段模型在 COCO 上的检测结果（单位：%）

Table4.4 The results of some one-stage models on the COCO data set(in: %)

Method	Backbone	Resolution	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SSD300	VGG-16	300x300	46	25.1	43.1	25.8	6.6	25.9	41.4
SSD321	Residual-101	321x321	11.2	28.0	45.4	29.3	6.2	28.3	49.3
DSSD321	Residual-101	321x321	9.5	28.0	46.1	29.2	7.4	28.1	47.6

SSD512	VGG-16	512x512	19	28.8	48.5	30.3	10.9	31.8	43.5
SSD513	ResNet-101	513x513	6.8	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101	513x513	5.5	33.2	53.3	35.2	13.0	35.4	51.1
STDN513	DenseNet-169	513x513	28	31.8	51.0	33.6	14.4	36.1	43.4
YOLOv3	Darknet-53	320x320	47	28.2	51.5	29.7	11.9	30.6	43.4
YOLOv3	Darknet-53	416x416	36	31.0	55.3	32.3	15.2	33.2	42.8
YOLOv3	Darknet-53	608x608	23	33.0	57.9	34.4	18.3	35.4	41.9

从表中结果可以看出，YOLOv3 与其它单阶段算法在相近的输入分辨率下，检测速度和检测精度均能做到较好的均衡；因此，本文考虑选用 YOLOv3 单阶段算法模型在智能平台上进行部署实现。

4.3　R-FCN 目标检测算法

4.3.1　算法原理

两阶段的 Faster R-CNN 在输入图像的特征图上提取区域特征，实现了特征提取操作在算法不同区域之间的共享。但是，在区域分类阶段，全连接的计算方式使得分类过程与前面大块的卷积网络相分离，无法共享计算；此外，全连接层针对大量候选区域特征的计算过程，计算量庞大。在此种情况下 R-FCN 网络应用而生，其整体的网络框架如图 4.3(a)所示，将 Faster R-CNN 中复杂的全连接部分替换为了卷积层，并将该部分卷积层置于了前端的特征提取部分，使得网络共享计算的范围得到了进一步的扩大。

整个算法模型主要由候选框生成和区域分类两大部分组成。在候选框生成部分，R-FCN 通过虚线框中的特征提取网络和 RPN，得到高召回率的候选区域；在区域分类部分，通过 1×1 的卷积对不同目标类别的相对位置信息进行编码，得到图中彩色的位置敏感得分图，并在结合了候选区域后，按照图 4.3(b)核心部分的流程，通过位置敏感 ROI 池化层（PSROIPool）对目标区域中的九个不同位置执行均值化操作，提取目标物体的空间区域特征，在获得了不同目标的空间信息后，对每一类的特征图按照加权平均的方式进行投票，并对投票结果执行 softmax，获得物体属于不同类的概率。

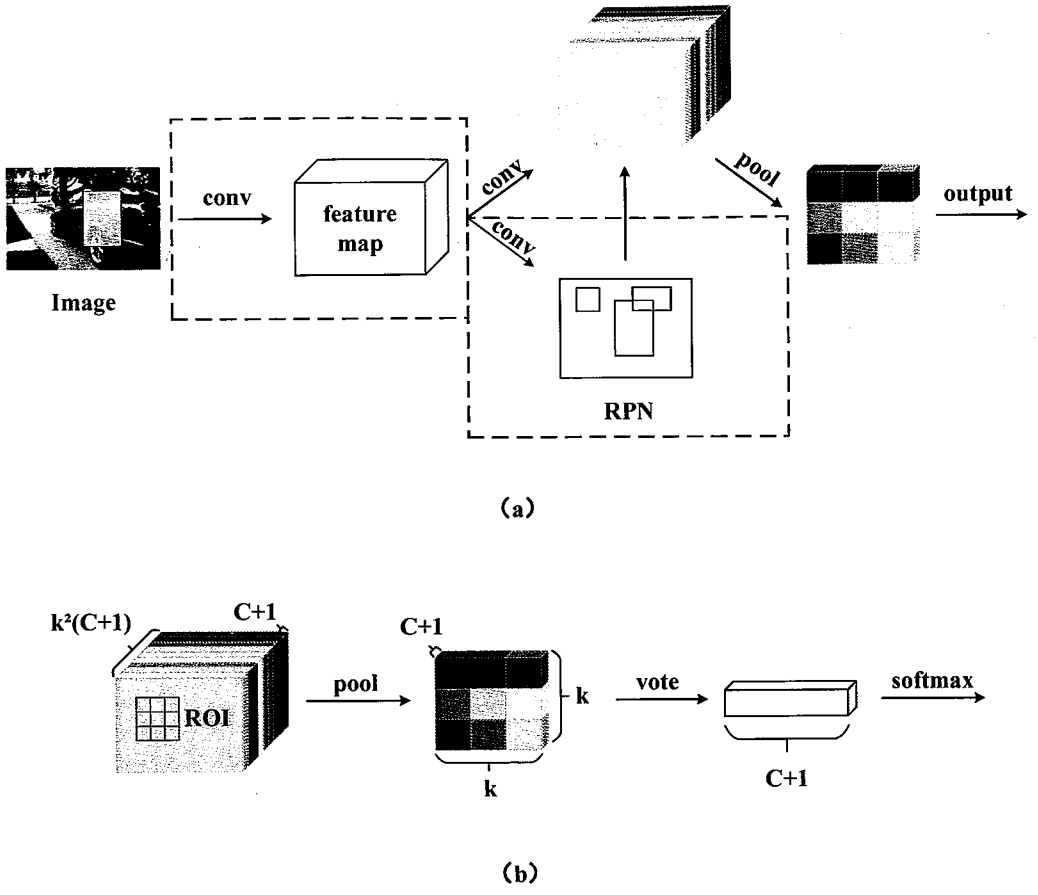


图 4.3　R-FCN 算法的网络框架

Figure4.3　The network framework of r-fcn algorithm

注：(a).整体网络框架 (b).核心部分

Note：(a).Overall network framework (b).Core part

通过前文算法选取阶段的检测对比结果，基于 ResNet50 的 R-FCN 网络相比其它两阶段算法在精度上依然存在优势，除此之外，对比 ResNet101 的骨干结构，ResNet50 在计算复杂度和模型大小上都更加符合边缘平台的部署；因此，本文考虑选用基于 ResNet50 的 R-FCN 作为智能平台最终的两阶段目标检测算法模型。

4.3.2　算法仿真

在 VOC 数据集上，对 BackBone 为 ResNet50 的 R-FCN 算法模型进行了训练。训练过程的动量参数是 0.9，权重衰减配置为5×10^{-4}，在每一次迭代中，随机将图片归一化为(400, 500, 600, 700, 800)的尺寸大小，进行多尺度的训练。此

外，还采用了困难样本挖掘的方法，即通过统计输入图片每个候选框的损失大小，排序选取损失最大的几个 ROI 进行训练；从而使基于 RseNet50 的 R-FCN 模型在 VOC 的测试集上得到前文表格所示的检测精度。本文选用的 R-FCN 算法基于 Caffe 架构，因此可以按照第 2 章 NNIE 的开发流程对 R-FCN 算法进行模拟仿真。整个仿真过程主要通过使用章节 3 中配置安装在 Windows 工作机上的 RuyiStudio 软件进行实现。

打开 RuyiStudio，创建 NNIE 工程，在工程配置界面中选择 Hi3559AV100 芯片和 MinGW GCC 编译链。创建完成后，会得到如图 4.4 所示的工程目录。其中，在 Caffe 模型通过 NNIE mapper 工具进行转换的过程中，Includes 包含了转换所需要的头文件，temp 内存储了转换前准备阶段产生的输出文件，转换完成后得到的输出文件则存放在 mapper 文件夹下，而 sim_out 主要用于保存数据指令文件在仿真库上仿真输出的内容，最后的 RFCN-ResNet50.cfg 文件，主要用于对 mapper 工具的转换过程进行配置。

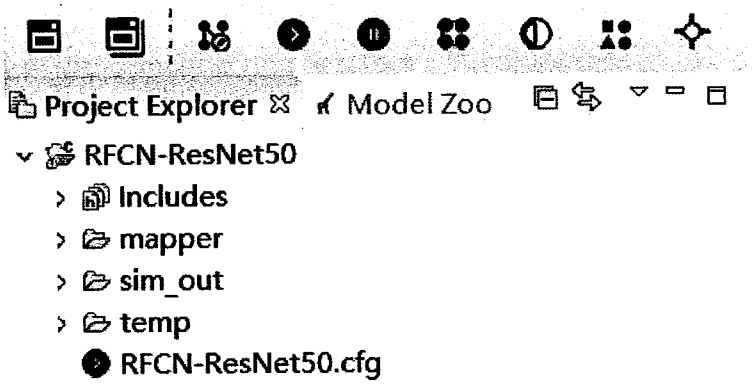


图 4.4　新建 NNIE 工程后的文件目录

Figure4.4　The file directory after creating NNIE project

在 mapper 工具转换算法模型的过程中，需要对 cfg 文件的部分选项进行编辑。在 cfg 的文本编辑模式下，为了使得 mapper 工具能够找到 Caffe 模型的网络结构描述文件，需要在第一个参数 prototxt_file 后面，输入 Caffe 模型对应的 prototxt 文件的相对路径，而对应的 caffemodel 文件相对路径，则添加在 caffemodel_file 参数后面；在 net_type 参数后，通过输入数字 0、1 或者 2，来指

定网络模型的网络类型，其中，0 对应 CNN 类型的网络，1 对应 ROI/PSROI 的网络类型，2 对应 LSTM、RNN 等 Recurrent 类型的网络；参数 log_level 后，可设置日志文件的输出及打印等级，等级范围从 0 到 3，0 级对应基本的函数和文件信息打印，1 级对应 mapper 工具解析文件信息的输出，2 级打印模型转换过程的中间信息，3 级打印最为详细的转换信息，输出文件量较大；在第五个参数 batch_num 后，可输入 0 到 256 范围内的整数，0 或 1 对应 mapper 工具单张图片的处理模式，其它整数则对应多张处理模式，在多张模式下，计算全连接会同时处理多张图片，提高计算资源的利用率；compile_mode 参数可设置编译的精度，配置范围从 0 到 2，有三个精度等级，在参数 0 下，对应低精度模式，会在转换过程中使用量化算法对 Caffe 模型的参数和数据进行量化，有利于生成较小的数据指令文件，但转换后的指令文件在精度上会有些许的损失，参数 1 对应高精度模式，生成的指令文件精度较好，但在整体的性能上会产生下降的现象，2 对应用户自定义模式，可根据需求个性化配置模型不同层的精度；internal_stride 用于配置模型转换过程中，中间结果在 DDR 颗粒上的对齐方式，能够提高内存颗粒的读写效率；sparse_rate 为稀疏率，用于稀疏网络参数提高模型的压缩率，稀疏值越高，模型的压缩率也就越高，例如，在配置为 0.5 的情况下，会通过参数压缩技术将 FC 参数的 50%稀疏为数值 0。

is_simulation 用于指定算法模型的转换类型，0 对应生成平台芯片上加载执行的数据指令文件，1 对应仿真模式，生成的指令文件可以通过 RuyiStudio 工具进行模拟仿真，在本文的算法仿真阶段，就采用的该种模式；instruction_name 参数用于重命名转换后模型文件的名称和文件存储的绝对路径；norm_type 参数可设置网络输入数据的预处理方式，与网络输入数据 image_type 配合使用，0 表示不做处理，在 image_type 输入为向量类型或者 NNIE 中 SVP_BLOB_TYPE_S32 类型 [97]时使用，3 表示对图像像素值进行缩放，参数 1 和 4 分别对网络数据执行减图像均值和减均值后再乘积的操作，此时，则需要在 mean_file 中输入对应均值文件的相对路径，2、5 参数表示减数据通道均值和减通道均值再乘积，同样的，在 mean_file 中输入通道均值文件的相对路径；image_list 用于保存 mapper 工具在量化过程中参考的图像或特征图列表的相对路径；RGB_order 包含 RGB 和 BGR 两个参数，用于指定输入图像的格式；max_roi_frame_cnt 用于指定 ROI

或 PSROI 所在网络模型中[31]，RPN 子网络输出的候选框数目，可接受的输入范围为[1, 5000]，在本文 R-FCN 网络的转换过程中，该参数设置为 300。

配置好 cfg 文件后，执行 make wk 操作，即可在 instruction_name 指定的目录下生成转换后的数据指令文件。由于 R-FCN 算法模型包含 NNIE 硬件单元不支持的 Non-support 层，因此，其网络结构在模型转换过程中，并不会全部交由 NNIE 硬件单元推理执行，所以，需要将 R-FCN 的网络结构进行分段，将网络的执行流程切分为 NNIE 和非 NNIE 两种不同的方式，其中，非 NNIE 执行的计算内容可通过调用 CPU 或者 DSP 硬件单元来实现。最终，R-FCN 网络模型结构的划分结果如图 4.5 所示。

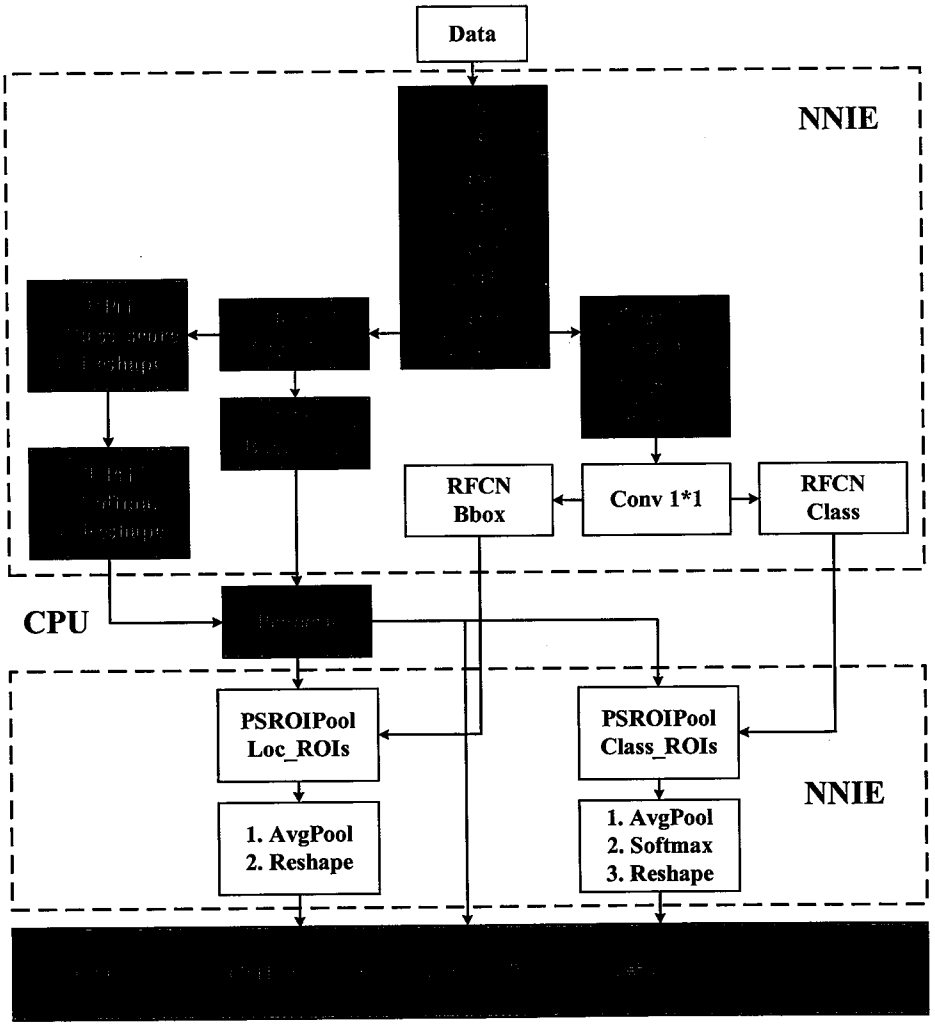
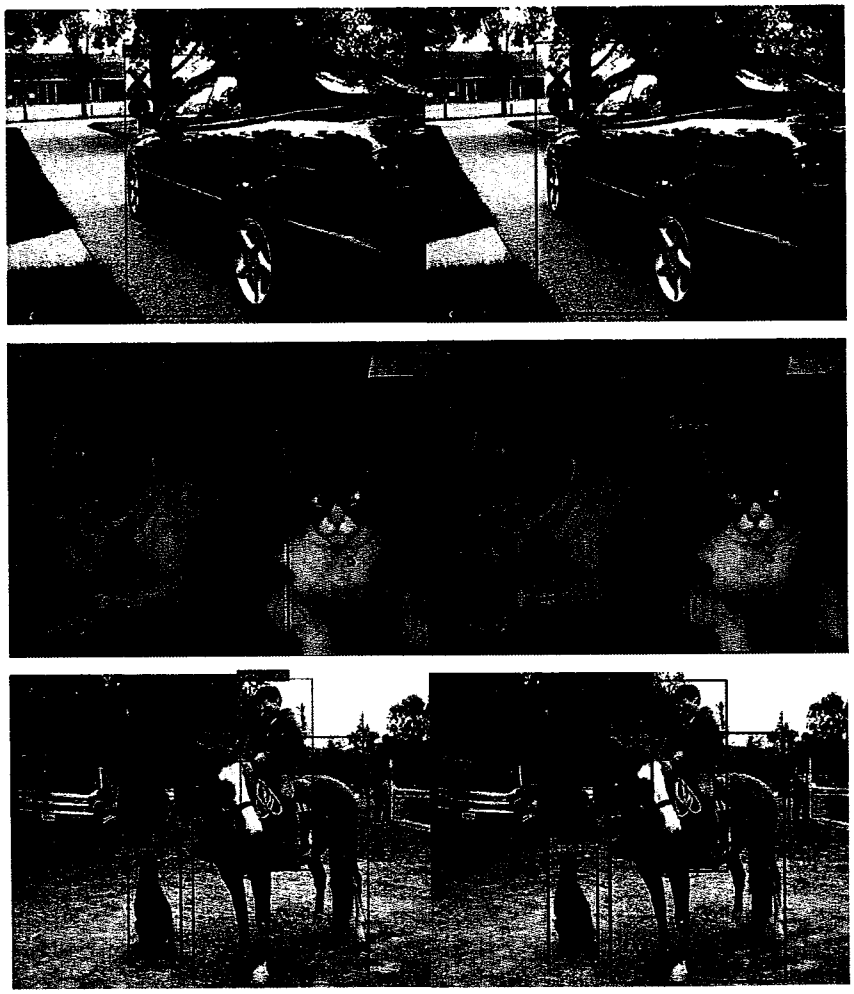


图 4.5　R-FCN 网络结构划分结果

Figure4.5　The results of the division of the R-FCN network structure

网络结构中被虚线框框起来的部分由 NNIE 硬件单元执行，被填充为红色的计算步骤，则通过调用 CPU 来实现。因此，R-FCN 网络模型的主要流程为 NNIE、CPU、NNIE 和 CPU 的分段执行方式；第一个 NNIE 执行阶段，主要由图 4.5 中填充蓝色的 ResNet[26]卷积网络和标注绿色的部分 RPN 网络结构组成，接下来的 CPU 执行阶段，主要计算 RPN 网络中生成矩形框的 Proposal 部分，对 RPN 前面得到的边框偏移信息和正负样本（前景和背景）类别进行综合，通过 IOU 和 NMS 等操作剔除过小和超出图像边界的候选区域，得到处理后的候选信息；随后的 NNIE 阶段，执行图 4.5 中标注为黄色的 PSROIPool 和后续的池化等操作；最后的 CPU 部分，对前面 NNIE 推理得到的结果执行后续的计算处理，可获得最终的候选框和类别置信度信息。



(a) (b)

图 4.6 R-FCN 网络模型的对比检测结果

Figure4.6 Comparative test results of R-FCN network model

注：(a).Caffe 模型下的检测结果 (b).仿真平台上的检测结果

Note：(a).Detection results under the Darknet framework (b).Detection results on the simulation platform

通过 mapper 工具得到了数据指令文件后，按照 R-FCN 网络模型被划分的执行方式，在 RuyiStudio 中编写对应的处理代码，编译生成 C++可执行文件，在仿真库上仿真执行。R-FCN 算法在 Caffe 模型下和仿真的部分对比检测结果如图 4.6 所示，左右两侧分别为 Caffe 模型和仿真的检测结果，从检测结果可以看出，右侧转换后的模型在第一行的大尺度目标、第二行的中尺度目标和最后的多尺度目标上，都能获得较好的检测结果。详细的检测精度如表 4.5 所示。

表 4.5　R-FCN 算法 Caffe 模型与仿真结果的精度对比

Table4.5　Accuracy comparison between R-FCN Caffe model and simulation results

目标尺寸	类别	Caffe 框架的 R-FCN	平台仿真下的 R-FCN
大尺度	car	100%	99.8%
中尺度	dog	99.4%	98.7%
	cat	98.6%	97.5%
多尺度	car	99.7%	99.2%
	dog	98.9%	99.1%
	horse	99.6%	99.4%
	person on horseback	97.6%	96.5%
	person	97.6%	94.9%

4.3.3　智能平台部署

在 R-FCN 算法仿真完成后，将转换后的数据指令文件在基于海思的智能处理平台上进行部署实现。与算法模型在 RuyiStudio 软件上的仿真过程相比，目标平台上的实现方式略有不同；在 Windows 工作机上进行仿真的过程中，是通过读取图片数据对单幅图像进行检测和显示输出，而在智能平台的嵌入式边缘设备

上，主要按照第三章中目标平台的总体处理流程来对算法模型进行部署实现。模型在智能平台上部署的数据流程如图 4.7 所示。

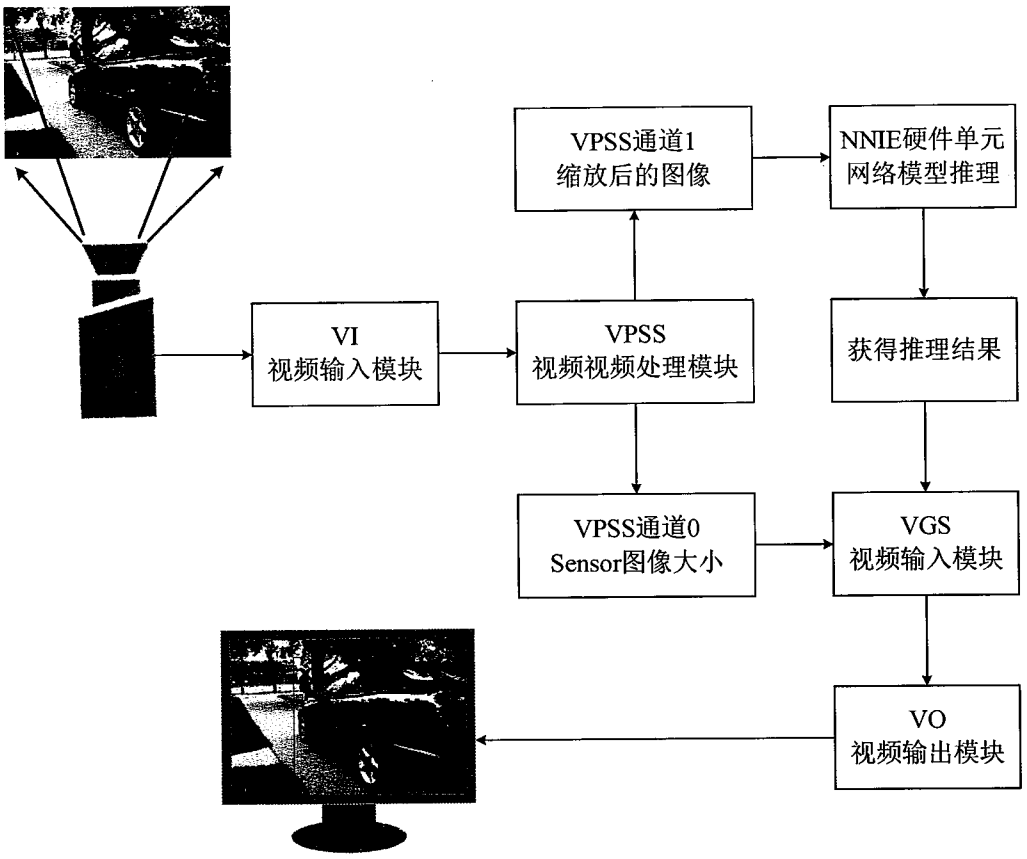


图 4.7　智能平台数据流程

Figure4.7　Data flow of intelligent platform

首先，通过图像采集部分的摄像头获取场景信息，将得到的数据传送给媒体处理平台的输入模块。VI 模块在获得摄像头捕获的 RAW 数据流后，对该数据进行去噪、色彩增强和强光抑制等初步的预处理，并将处理后的图像数据发送给 VPSS 模块。VPSS 在接收到数据后，通过执行帧率控制、动态对比度增强和尺寸缩放等操作，对图像信息进行进一步的处理；处理完成后，在 VPSS 组内的双通道中，将数据划分为两路，一路原始图像尺寸大小的数据在通道 0 内直接传送给视频图形系统 VGS，另一路经过缩放的图像在通道 1 内发送给 NNIE 硬件单元；在 NNIE 单元内部，按照算法仿真阶段 R-FCN 网络结构执行方式的划分结果，加载模型转换后的数据指令文件，对通道 1 内的图像进行推理检测，并将模型推

理后的检测结果发送给 VGS 模块；VGS 模块在接收到传来的信息后，将 NNIE 推理后的结果以矩形框的形式叠加在 VPSS 通道 0 内原始尺寸大小的图像数据上，最后通过媒体处理平台的输出模块，经由 HDMI 接口将检测后的结果在显示器上显示输出。

为了提高智能平台对图像原始 RAW 数据的传输和处理效率，本文在采集单元的 VI 模块中设计了一种基于单芯片的并行处理架构 [98]，在该架构基础上采用双图像信号处理（ISP）单元，对采集得到的大量时序数据进行并行处理。该并行处理架构的整体处理步骤如图 4.8 所示。

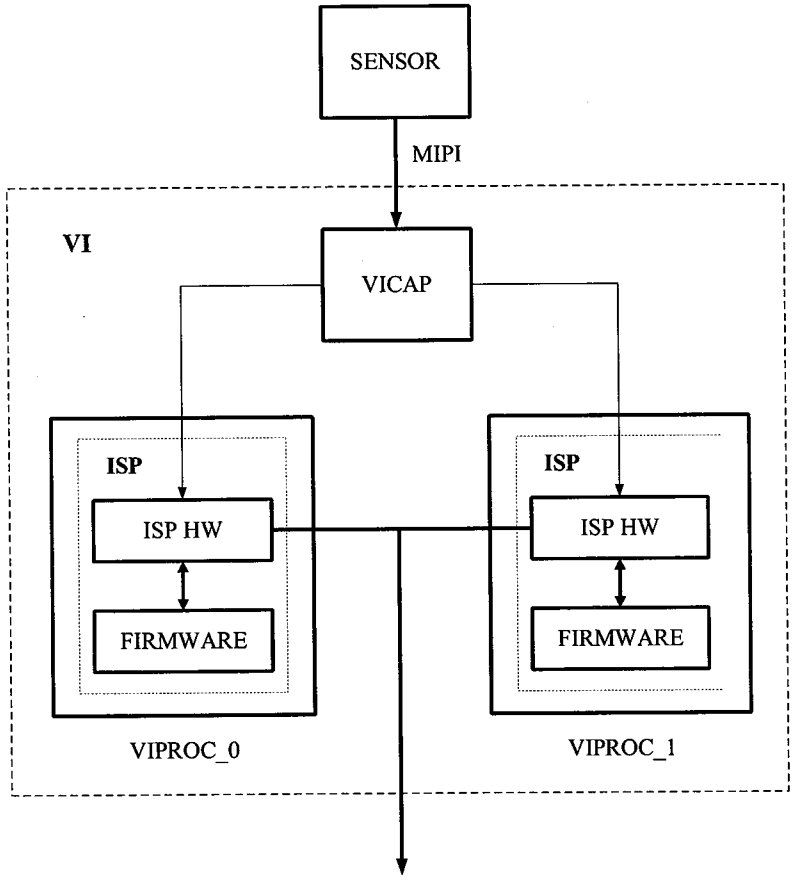


图 4.8　VI 模块内并行架构处理流程

Figure4.8　The processing flow of the parallel processing architecture in the VI module

在该并行架构下，VI 模块通过 VICAP 捕获智能平台图像传感器采集的数据流，捕获成功后，VICAP 将捕获到的图像帧数据直接传送到两个处理子模块 VIPROC 中进行并行处理。在 VIPROC_0 和 VIPROC_1 内，通过内嵌的双 ISP 单元，对接收到的 RAW 数据进行处理。其中，在 ISP 内部，主要由 ISP 硬件逻辑

ISP HW 和固件单元 FIRMWARE 两部分组成。FIRMWARE 固件通过获取 ISP HW 对当前图像帧数据的实时统计结果，反馈调节 ISP 的硬件逻辑部分；ISP HW 在接收到反馈信息后，通过执行相应的图像处理算法，实现图像效果的调节；ISP 单元内部在该反馈控制机制的作用下，即可实现图像质量的自动调节。在 VI 模块对数据初步处理完成后，VIPROC 中的数据将直接传送给 VPSS 进行处理。

在上述的处理流程下，同时省去了第一阶段 VICAP 写出数据到内存，VIPROC 从内存中获取数据，第二阶段 VIPROC 写出初步处理后的数据到 DDR，VPSS 从 DDR 读出数据执行后处理的过程。相比传统的处理流程，节省了对 DDR 内部结构充放电，进行调用的中间过程，在加快了智能平台数据传输和处理速度的同时，达到了低功耗的目的。

转换后的 R-FCN 算法在智能平台上可实现数据输入、模型推理和结果输出的一体化目标检测。在平台数据处理过程用到的 VI、VPSS 和 NNIE 等模块中，NNIE 智能推理单元起到了至关重要的作用，其在智能平台中执行 R-FCN 数据指令文件的软件过程和其在平台中的逻辑位置[21]分别如图 4.9 和 4.10 所示。

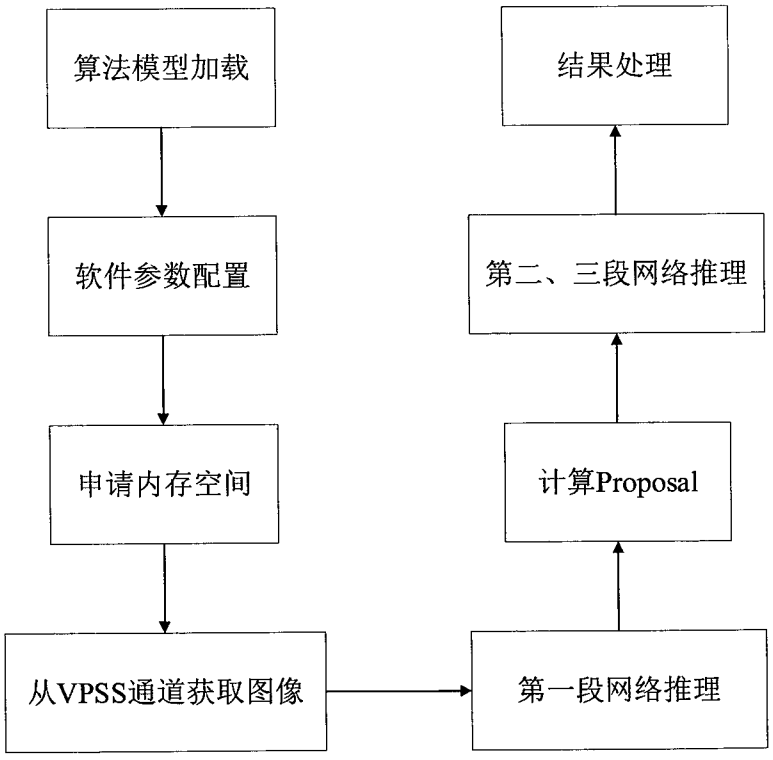


图 4.9　R-FCN 指令文件在 NNIE 中的软件处理流程

Figure4.9　Software processing flow of R-FCN instruction file in NNIE

在媒体处理平台的 VI 和 VPSS 模块对采集到的图像信息处理完成后，NNIE 硬件单元首先对转换后的算法模型进行加载，加载完成后，对 NNIE 和 R-FCN 网络模型的软件参数进行配置，并根据设置参数的大小，在平台的内存上申请所需的临时存储空间。在 NNIE 中，主要对推理图片的尺寸大小、NNIE 单次处理图片数和最大 ROI 等参数进行配置，并根据前文 R-FCN 网络结构执行方式的划分结果，在具体的推理流程中，将 R-FCN 转换后的算法模型分成了 3 段；第一段主要为网络划分结果中第一个 NNIE 执行阶段的网络结构，第二段和第三段则分别由第二个 NNIE 执行阶段中的 PSROIPool_Class_ROIs、PSROIPool_Loc_ROIs 和它们对应的后续操作过程组成。在 R-FCN 的软件参数部分，包括 RPN 中 9 种 Anchor 的基本尺寸大小和长宽比、非极大抑制的阈值和最低置信度等。

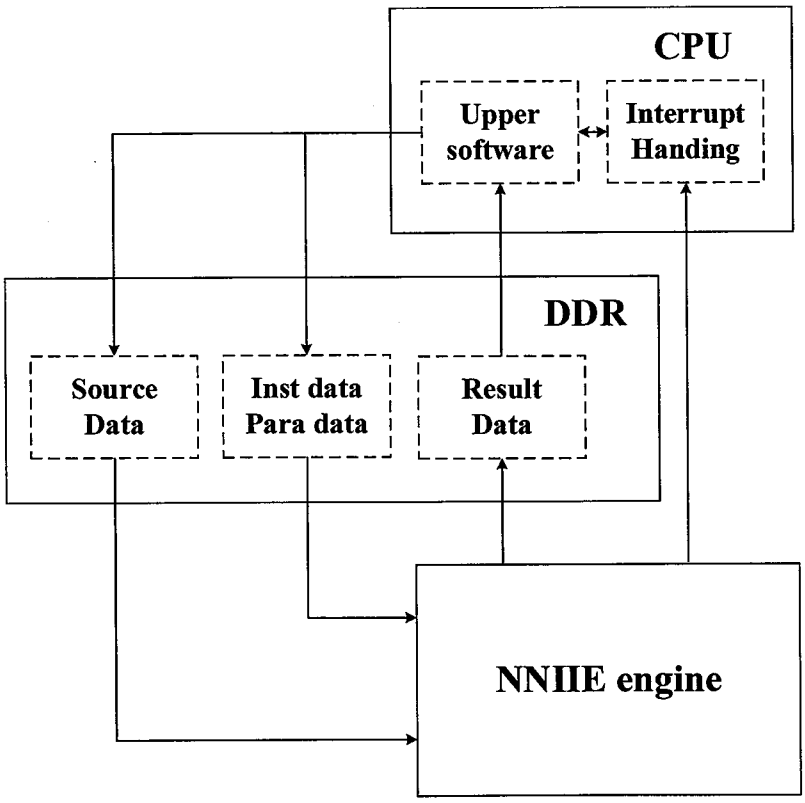


图 4.10　NNIE 在智能平台中的逻辑位置

Figure4.10　The logical position of NNIE in the intelligent platform

在整体的参数设置和空间分配完成后，从 VPSS 的通道中取出 VI 和 VPSS 处理后的图像数据，并将取得的数据送入 R-FCN 算法模型的第一段，调用 NNIE

硬件单元对该段网络执行前向推理。在第一段网络推理过程结束后，按照图 4.10 的逻辑结构，调用 CPU，执行图 4.5 网络划分中的 Proposal 部分，对第一段网络的推理结果进行处理；待 CPU 处理完成后，按照 R-FCN 算法的流程，将 CPU 处理后的结果送入 NNIE，在 NNIE 中对后续的第二和第三段网络进行推理。待得到第二和第三段网络的处理结果后，再次调用 CPU，在 BBox tranform 和 Clip bbox 等操作的 C 代码下，对该结果进行二次处理，并将处理后的结果发送给 VGS，用于绘制后续的目标检测框。

4.4 YOLOv3 目标检测算法

在两阶段的 R-FCN 算法模型仿真和部署完成后，选用单阶段的 YOLOv3 网络模型执行同样的操作，考证单阶段目标检测网络在国产智能处理平台上部署实现的可行性。

4.4.1 算法原理

YOLOv3 算法的网络框架如图 4.11 所示。该网络通过一个端到端的全卷积模型，实现了图像的特征提取、分类和定位，相比两阶段的检测模型，更加的简洁和高效。YOLOv3 以 Darknet53 作为骨干网络，该骨干网络主体主要由 1×1 和 3×3 的小型卷积核堆叠而成，结构简单高效，在对图像进行特征提取的过程中，采用了多级的残差模块，在增强了 Backbone 学习能力的同时，减小了网络产生梯度爆炸现象的风险。YOLOv3 网络模型在最后的输出阶段，融合了模型不同层次的语义信息，在不同大小的特征图上分别检测不同尺度的目标物体。以 416×416 大小的输入图像为例，Backbone 网络在获取图像后，分别提取得到了 52×52、26×26 和 13×13 三种不同尺度的图像特征，之后，经过多层次的卷积操作，13×13 的特征图作为 yolo 检测层的输入，对输入图像中大尺度的目标物体进行检测，在这之后，13×13 的特征图还要经过两倍上采样，与 26×26 的特征图进行融合，融合结果作为检测层的输入，负责中尺度目标物体的检测，而余下的 52×52 的特征图则与前面融合结果的两倍上采样再次进行融合，并将再次融合后的结果输入到 yolo 检测层，实现输入图像中小尺度目标的检测。

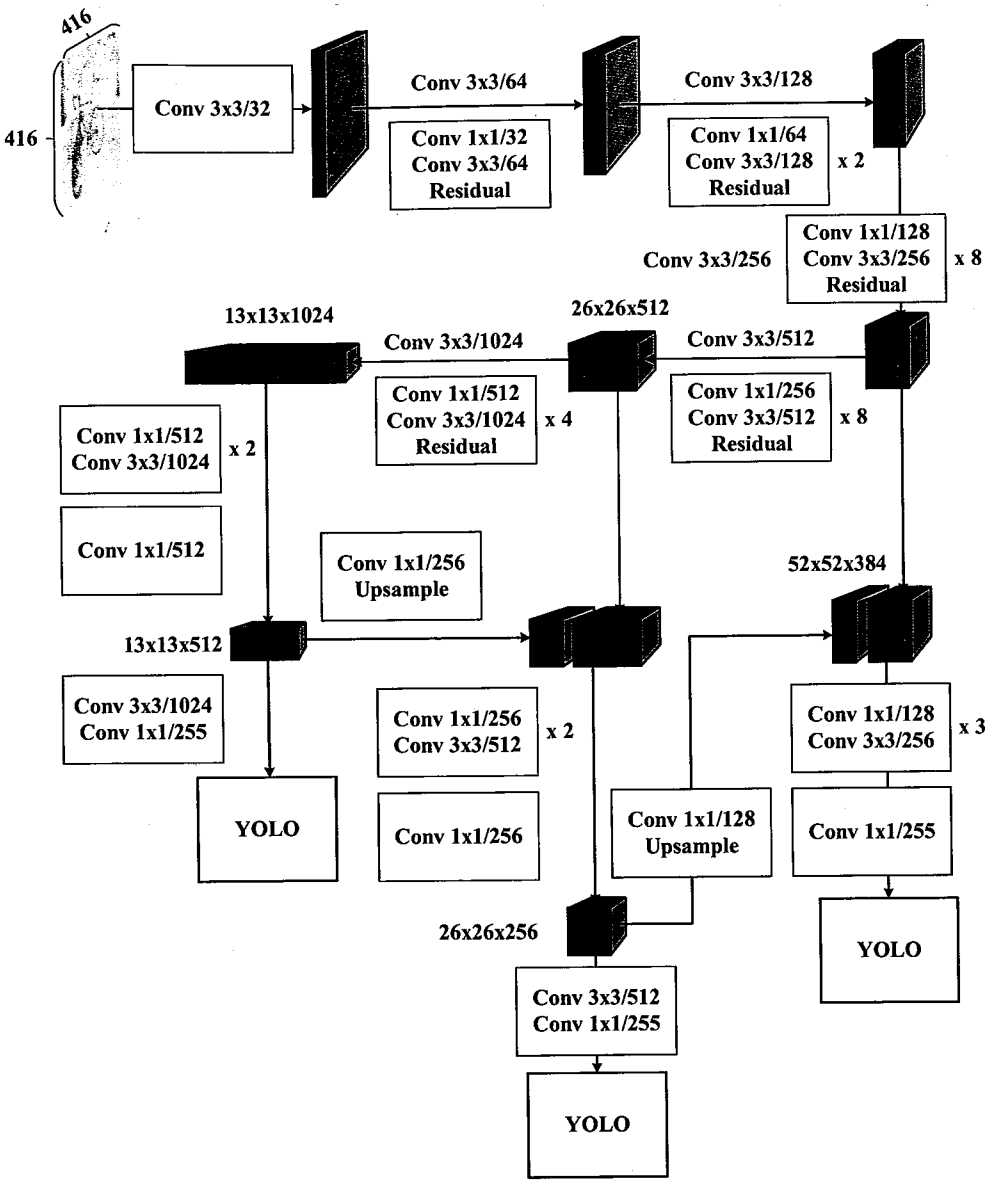


图 4.11　YOLOv3 网络框架示意图

Figure4.11　Schematic diagram of YOLOv3 network framework

4.4.2　模型转换

YOLOv3 的算法模型基于原生的 Darknet 框架，而当前的智能处理平台仅支持 Caffe 模型，因此，需要对 YOLOv3 的网络框架进行转换。整个转换流程如图 4.12 所示，主要是将 Darknet 下的*.cfg 网络描述文件和*.weight 权重，转化为 Caffe 框架下能够识别的*.prototxt 描述文件和权重*.caffemodel 文件。

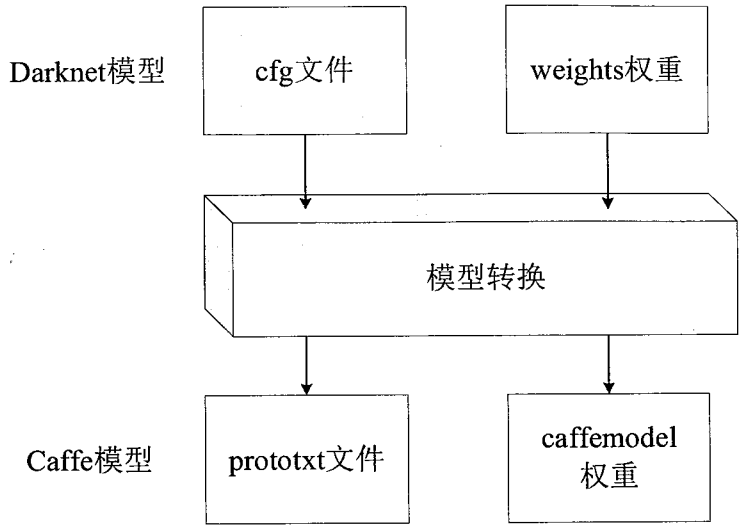


图 4.12　Darknet 框架到 Caffe 的转换流程

Figure4.12　Darknet framework to Caffe conversion process

基于 Darknet 的 YOLOv3 算法在转换为 Caffe 框架的过程中，会存在一些网络结构无法得到 Caffe 原生框架支持的情况。因此，需要在 Caffe 框架下，通过寻找替代层或者扩展原生层的方法来解决该问题。比如 YOLOv3 网络结构中的 route 和 shortcut 结构，在 Caffe 框架下，均没有对应的实现代码，无法得到 Caffe 框架的识别和支持；但是，在 Caffe 支持的网络结构中，concat 和 eltwise 操作则能够分别实现 YOLOv3 中 route 和 shortcut 类似的功能，因此，可以将 Caffe 不支持的 route 和 shortcut 层，分别替换为对应的 concat 和 eltwise 网络结构。

除了 YOLOv3 中的 route 和 shorcut 结构外，对特征图进行上采用操作的 Upsample 结构，也无法在 Caffe 的原生框架下正常运行，但不同与前两者，Upsample 无法在 Caffe 支持的网络结构中寻找替代操作，只能通过网络结构扩充的方式来解决。首先，修改 Caffe 源码中的 caffe.proto 文件，在 proto 文件中的 LayerParameter 信息下，添加 Upsample 层的参数声明，主要包括参数名称和字段等信息；完成 Upsample 参数在 proto 文件下的声明后，添加一个新的 UpsampleParameter message 信息在 caffe.proto 文件的末尾，该信息主要对 Upsample 参数所使用的数据类型、字段名称和默认情况下的使用数值等内容进行配置。在上述 proto 文件修改完成后，Upsample 网络结构就能够在 Caffe 框架下得到识别，但要具体实现 Upsample 结构的解析和推理等功能，还需要按照

Upsample 的处理方式，在 Caffe 源码的 layers 目录下添加底层代码的实现，根据执行单元的不同，需要分别添加能够在 CPU 和 GPU 上运行的 upsample_layer.cpp 和 upsample_layer.cu，在代码的具体实现过程中，主要对 Upsample 结构的前向推理和反向传播功能进行复现，并在代码的末尾引入前文 caffe.proto 文件中声明的 Upsample 层函数。本文在 github 开源工程的基础上，按照上述内容的实现要求，对 Upsample 网络结构的底层代码进行了实现。

在上述步骤都执行完毕后，对整个 Caffe 框架进行编译，编译成功后，在 Caffe 下就能够识别和解析原本无法支持的网络结构了。在 Darknet 到 Caffe 模型适配完成后，通过代码脚本对 YOLOv3 算法的网络描述和权重文件进行转换，即可得到 Caffe 框架下所需的 prototxt 和 caffemodel 文件。综上所述，YOLOv3 算法的 Darknet 模型到 Caffe 框架的整体适配如图 4.13 所示。

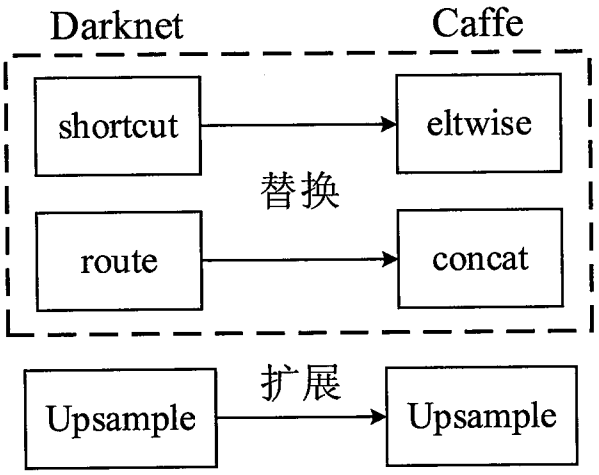


图 4.13　YOLOv3 算法的 Darknet 模型到 Caffe 框架的适配

Figure4.13　Adapation of the Darknet model of the YOLOv3 algorithm to the Caffe framework

4.4.3　算法仿真

在 YOLOv3 算法模型转换完成后，采用与前文 R-FCN 相似的方式，在 RuyiStudio 工具上创建新的 NNIE 工程，通过编辑 cfg 文件，对 mapper 工具的转换过程进行配置，在指定目录下生成 YOLOv3 模型对应的数据指令文件。

在得到的指令文件中，将 YOLOv3 算法模型的执行流程切分成了 NNIE 和 CPU 两个不同的阶段。YOLOv3 整体的网络结构，在具体推理过程中被划分的结果如图 4.14 所示。

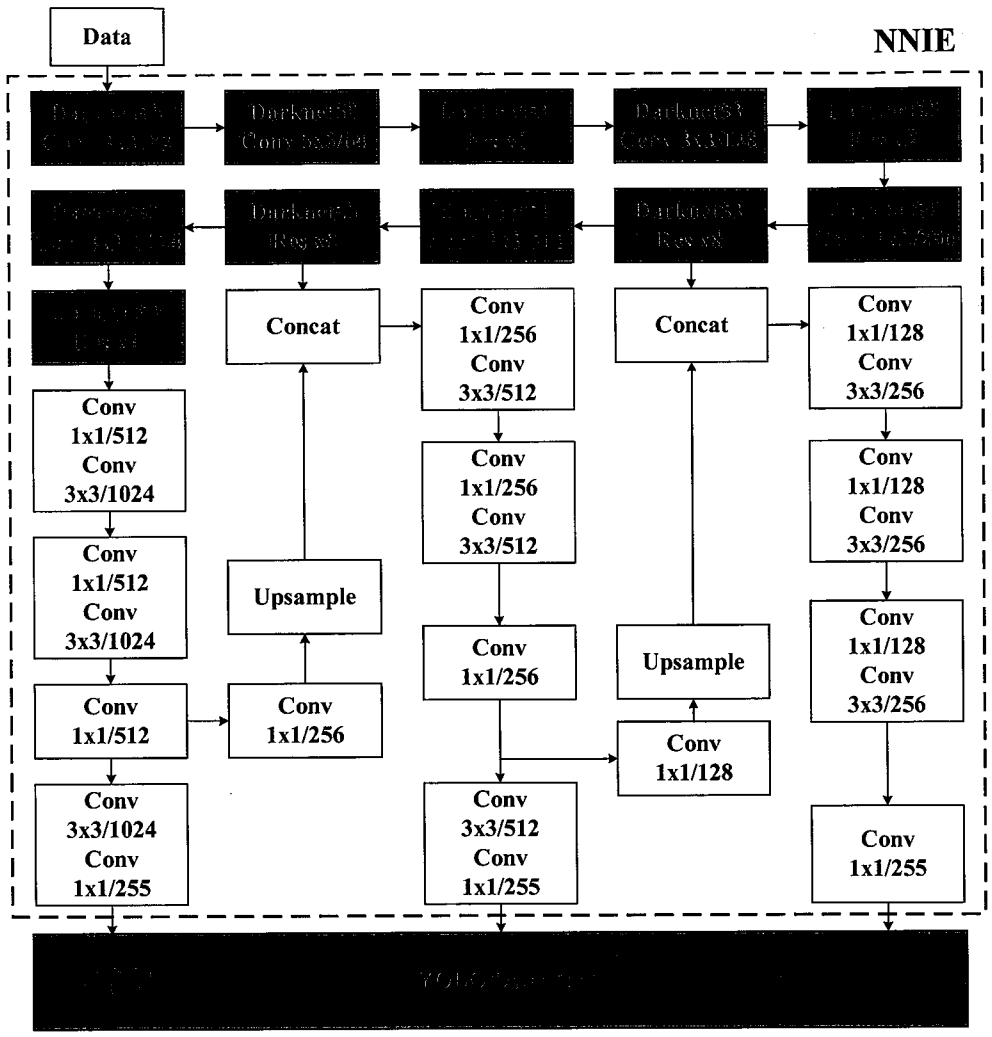
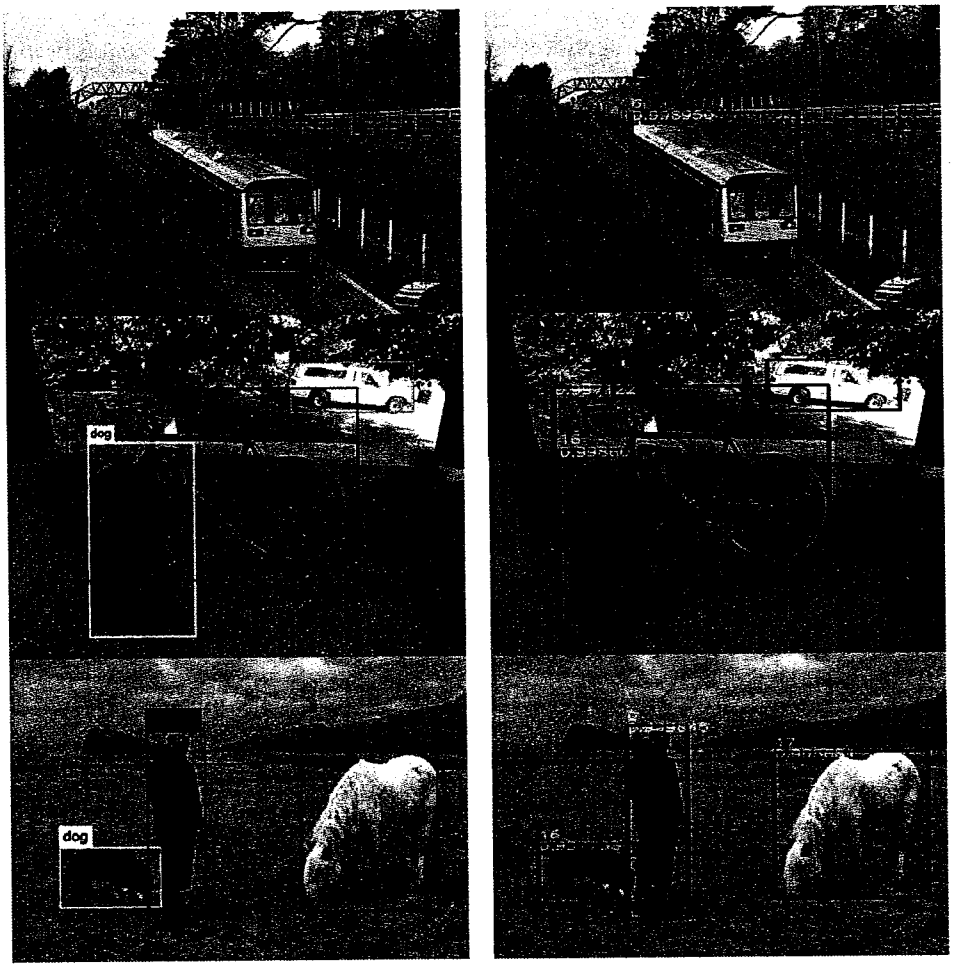


图 4.14　YOLOv3 网络结构的划分结果

Figure4.14　The results of the division of the YOLOv3 network structure

YOLOv3 的整个网络结构被划分成了两部分，虚线框框起来的第一部分由 NNIE 硬件单元执行，而填充红色的第二部分则由 CPU 硬件进行计算。在第一部分中，主要由填充蓝色的 Darknet53 骨干网络和卷积结构组成，该部分占据着 YOLOv3 网络模型中大部分的计算操作，可实现输入图像的特征提取和不同尺度特征的融合；在第一部分得到三种不同尺度的特征后，特征向量会被传送给 CPU 执行的第二部分，该部分主要由 YOLOv3 中的 yolo 检测层组成，该检测层在获得了第一部分的特征向量后，可通过解析该向量，得到最后的分类结果和边框信息。

在 RuyiStudio 中，按照图 4.14 中 YOLOv3 网络的执行流程，编写对应的 C++ 代码，在模拟仿真库上模拟调用 NNIE 和 CPU 硬件单元，对 YOLOv3 算法的前向推理过程进行实现。



(a)　　(b)

图 4.15　YOLOv3 网络模型的对比检测结果

Figure4.15　Comparative test results of YOLOv3 network model

注：(a).Darknet 框架下的检测结果 (b).仿真平台上的检测结果

Note： (a).Detection results under the Darknet framework (b).Detection results on the simulation platform

YOLOv3 网络模型在 Darknet 和仿真库上的部分对比检测结果如图 4.15 所示，左侧一栏为 Darknet 框架下的检测结果，右侧一栏为仿真库上的检测结果，

从检测结果可以看出，转换后与转换前的模型均能获得较好的检测效果。详细的检测精度对比如表 4.6 所示。

表 4.6　YOLOv3 Darknet 模型和仿真平台的精度对比

Table4.6　Accuracy comparison between YOLOv3 Darknet model and simulation platform

类别	Darknet 模型	仿真平台
train	100%	99%
dog	100%	99%
bicycle	99%	99%
truck	92%	87%
dog	99%	99%
horse	100%	99%
person	100%	99%

4.4.4　智能平台部署

在 YOLOv3 算法模型仿真的过程中，当转换后的模型在功能和精度上能够达到实际需求后，可将转换后的 YOLOv3 算法在智能平台上进行部署实现。YOLOv3 的实现流程和前文 R-FCN 网络模型在智能平台上的部署流程类似，同样是通过目标平台的摄像头采集场景信息，将采集得到图像数据经由媒体处理平台的 VI 和 VPSS 模块进行初步处理，并在 VPSS 的通道 1 中，调用智能平台的 NNIE 硬件单元进行检测，在得到检测结果后，传送给 VGS 单元；通过该单元执行目标检测框在图像上叠加的操作，待 VGS 处理完成后，通过平台的 VO 模块实现最终的输出显示。

在智能处理平台上，YOLOv3 算法模型的软件部署实现，需要遵照相应的处理步骤，主要包括硬件平台初始化、内存中临时缓存空间的配置和初始化以及媒体处理平台中各个模块的配置和初始化等。YOLOv3 算法在海思智能平台上部署实现的整体软件流程如图 4.16 所示，主要分为五个阶段。

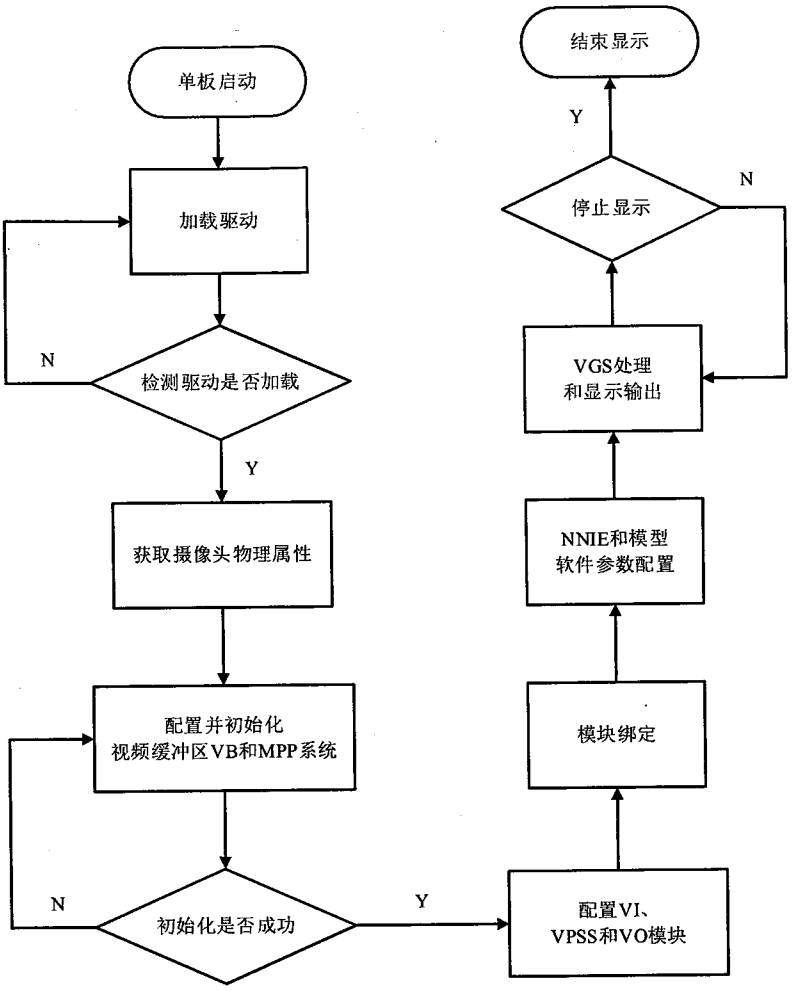


图 4.16　YOLOv3 算法的整体软件流程

Figure4.16　The overall software flow of the YOLOv3 algorithm

首先，在目标平台上电启动后，加载图像传感器、Hi3559AV100 核心芯片和平台其它外围硬件单元的驱动程序，并通过标志位判断驱动是否被正确加载执行。在平台硬件驱动检测通过后，获取图像传感器的物理参数，主要包括传感器的物理分辨率、图像像素格式和硬件传输接口信息等，并将获取的参数以结构体的形式在程序的全局变量区域内进行存储，方便后续处理模块的获取和配置。

第二阶段，在上述过程结束后，对媒体处理平台和其内部控制系统内存分配与回收过程的视频缓冲池 VB（Video Buffer）进行初始化。其中，VB 主要向媒体处理应用程序提供大容量物理内存空间的操作管理功能，通过配置内存空间连续物理地址上相同大小的缓存块，来构建缓冲池，使得不同的媒体处理单元能够合理有效的利用物理空间。由于平台系统在运行过程中，之前不同处理功能的应

用程序，可能使用了相同的功能模块，并对该模块的具体参数进行了适配；因此，在再次配置初始化之前，需要调用媒体处理软件平台的 HI_MPI_SYS_Exit 和 HI_MPI_VB_Exit，分别对 MPP 和视频缓冲池执行去初始化的操作，复位前序应用程序的配置。在复位操作处理完成后，根据图像传感器采集过程中所需的空间，调用 HI_MPI_VB_SetConfig 函数，对缓冲池的参数进行配置。之后，通过先后执行 HI_MPI_VB_Init 和 HI_MPI_SYS_Init 函数，实现缓冲池和媒体处理软件平台的初始化。第三阶段，在媒体处理平台内部，分别对 YOLOv3 算法部署实现过程中，与图像信息数据流相关的视频输入、视频处理和输出模块进行配置，并在 VI 和 VPSS 模块配置完成后，通过执行 HI_MPI_SYS_Bind，将 VI 管道与 VPSS 组内的通道进行绑定，实现图像数据流在两个不同处理单元间的自动传递，从而提高系统整体的处理速度[98]。

第四阶段，对目标平台的核心智能推理单元 NNIE 进行配置。首先，通过 fopen 等文件操作函数，获取 YOLOv3 算法模型对应数据指令文件的大小和分段信息等基本参数，随后通过媒体处理平台的 MPI 函数 HI_MPI_SYS_MmzAlloc，在智能平台上按照获取的参数信息开辟内存缓存空间，为后续正式的模型加载工作做准备。在存储空间开辟完成后，对包含 NNIE 参数的全局结构体变量进行配置，类似前文 R-FCN 算法模型在部署阶段的 NNIE 配置，主要包括待推理模型的网络分段信息、检测图像的尺寸和数目等；其中，在 YOLOv3 算法模型的部署中，网络仅被划分了单独的一段，该部分主要由 YOLOv3 仿真阶段，网络结构划分结果中第一部分的 Darknet53 和其余的卷积网络结构组成。至此， NNIE 模型加载前的空间分配和参数配置基本完成。接下来，对 YOLOv3 算法模型的软件参数进行配置，主要包括检测类别数、三个 YOLO 检测层所需要的特征图大小以及九组 Anchor 对应的尺寸信息等。当 YOLOv3 的 NNIE 和网络模型的软件参数配置完成后，载入 YOLOv3 转换后的数据指令文件和 VPSS 通道中的待检测图像到 NNIE 硬件单元，经由 YOLOv3 网络划分的 NNIE 部分，对图像数据进行推理，实现目标图像的特征提取和融合操作；在获得推理结果后，将得到的特征结果传递给 CPU 硬件单元进行后处理，实现 YOLOv3 网络中 YOLO 检测层的功能，并将解析后的结果传递给后续的 VGS 模块。

第五阶段，在 VGS 模块接收到最终的推理结果后，首先通过媒体处理平台的 HI_MPI_VGS_BeginJob 函数，启动一个绘制目标检测框的 job，之后，对检测

框的颜色和样式进行配置，配置完成后，根据检测结果的目标数，循环调用 HI_MPI_VGS_AddCoverTask，往已经启动的绘框 job 中添加配置后的检测框属性，并通过 HI_MPI_VGS_EndJob 提交该 job，执行最终的检测框绘制操作。最后，将叠加候选框后的数据传递给 VO 模块进行输出显示 [23]。

4.5　实验对比和结果

前文 R-FCN 和 YOLOv3 算法模型部署完成后，在 changedetection 2014 的数据集 [99]上分别对其进行了测试。

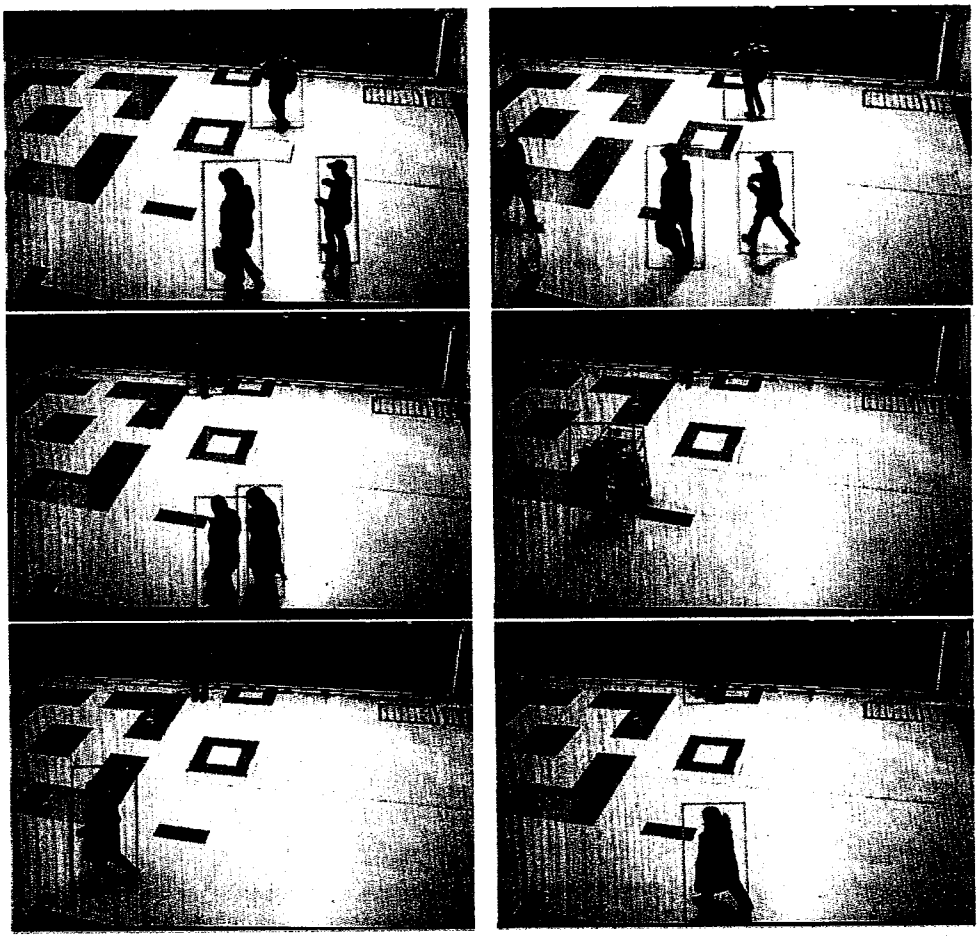


图 4.17　R-FCN 部署检测结果

Figure4.17　R-FCN deployment test results

由于智能平台直接通过连接的显示器进行检测结果的实时输出，所以本文采用摄像设备，对显示器中输出的检测结果进行了录制。图 4.17 和 4.18 分别为 R-FCN 和 YOLOv3 算法在智能检测平台上，对 pedestrians 连续视频检测输出结果

录制视频中的部分关键帧。由于平台镜头焦距的原因，所检测到的画幅相比原数据略小，此外，结果的关键帧也因摄像设备录制显示器的原因，存在些许的彩色条纹。

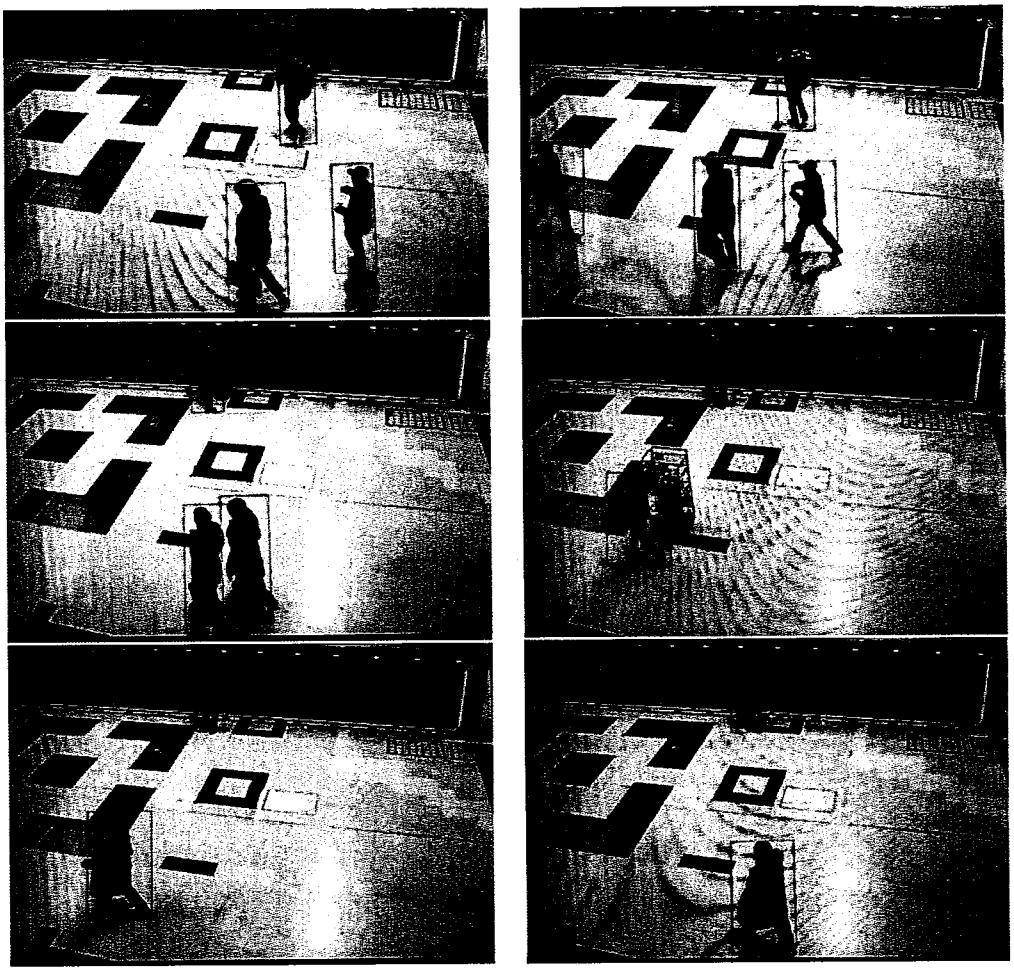


图 4.18　YOLOv3 部署检测结果

Figure4.18　YOLOv3 deployment test results

从检测结果可以看出，R-FCN 对视频中的主体目标都能够做到有效的检测，但对比 YOLOv3 算法，R-FCN 在目标对象的框取范围上略大，此外，在对移动目标的检测过程中，检测结果也不太稳定，例如，视频中最上方的背包男子，在 R-FCN 第二行的第二个和第三行的第一个检测结果中，目标框均产生了不同程度的抖动和缩小，而在 YOLOv3 算法的检测结果中，对应的检测结果都比较稳定，检测框的方位也更加的贴合目标对象。因此，在目标平台上，与基于 ResNet-50 骨干网络的 R-FCN 相比，YOLOv3 的最终检测效果相对较好。

尽管 R-FCN 属于两阶段的算法模型，但是本文选用的骨干网络 ResNet-50，相比于单阶段模型 YOLOv3 采用的 Darknet-53，在模型提取能力上还存在一定的不足。如表 4.7 所示，在 ImageNet 数据集 Top1 和 Top5 的测试结果中，Darknet-53 的检测精度均要高于 ResNet-50。

表 4.7　Darknet-53 和 ResNet-50 骨干网络的对比结果

Table4.7　Comparison of Darknet-53 and ResNet-50 backbone network

BackBone	Parameters	Top-1	Top-5	Bn Ops
Darknet-53	40.5×10^6	77.2%	93.8%	18.7
ResNet-50	25.5×10^6	75.8%	92.87%	9.74

在检测速度方面，分别对比了单芯片并行处理架构应用前后的检测帧率和 YOLOv3 与 R-FCN 在 NNIE 上的具体推理时间。

表 4.8　单芯片并行处理架构应用的帧率对比

Table4.8　Single-chip parallel processing architecture before and after frame rate comparison

算法模型	并行架构应用前的 FPS	应用后的 FPS
YOLOv3	7.3	9
R-FCN	11	13

表 4.9　YOLOv3 和 R-FCN 在智能平台上的速度对比

Table4.9　Speed comparison of YOLOv3 and R-FCN on smart platform

算法模型	BackBone	NNIE 推理时间	FPS
YOLOv3	Darknet-53	107ms	9
R-FCN	ResNet-50	73ms	13

从表 4.8 并行架构应用前后的对比结果中可以看出，在并行架构的作用下，YOLOv3 和 R-FCN 的帧率都提升了 2 帧左右。由于 YOLOv3 的 Darknet-53 在表 4.7 所示的参数量和计算操作数上都要高于 R-FCN 的 ResNet-50，因此，如表 4.9

所示，R-FCN 在平台的推理时间和帧率上具有些许的优势。其中，帧率较低的 YOLOv3 算法模型，相比其在国外 TX2 平台（基于 GPU 架构）上的 5.3 FPS，具备有一定的优势。

4.6 本章小结

本章主要在国产的智能处理平台下，对基于深度学习的目标检测算法进行了部署研究。首先围绕目标检测的传统方法和基于深度神经网络的方法进行展开，阐明了从传统方法到深度学习方法的发展过程；接下来，对基于深度算法的两阶段模型和单阶段模型进行了简单介绍，并对比选取了两阶段的 R-FCN 和单阶段的 YOLOv3 算法作为最终的部署模型，紧接着，对其各自的算法原理、仿真验证、网络模型的协同转换和平台有限资源下的协同设计部署进行了详细的阐述，并在实际的部署过程中，基于该国产平台设计了一种并行处理架构，提高了平台整体的处理速度；最后，在公开数据集上，分别对部署后的算法模型进行了对比测试。

第 5 章　智能平台的无人机检测应用

5.1　需求分析

近年来，无人机以其灵活、便携的特点，在农业、工业和影视创作等领域得到了广泛的应用，并随着应用场景的变化，拓展出了各种各样的使用功能。但在无人机不断丰富和发展的过程中，其带来的安全隐患也日益凸显，因此，研发无人机的实时检测平台，符合现实场景中的应用和发展需要。本文在国产智能处理平台的基础上，针对实验室项目的实际需求，对无人机的目标检测进行了部署和实现。

在智能平台的算法部分，考虑到无人机检测的实时性，基于深度学习的单阶段算法模型能够较好的满足应用需求。目前，在单阶段算法模型中，SSD 在无人机检测领域已经得到了应用，包括在 SSD 算法的基础上增加 ResNet 分类网络和 KNN 算法，对目标图像无人机进行特征提取和分类，以及通过 SSD 和 AlexNet 模型，检测微调图像中的目标无人机等方法，但是，基于 SSD 改进的目标检测算法，整体的处理步骤较多 [100]，在嵌入式平台上的处理速度较慢，因此，本文考虑采用单阶段的 YOLOv3 系列算法模型。

5.2　模型优化与仿真

YOLOv3 系列算法包括 YOLOv3 和 YOLOv3-tiny 两种。为了对比两者的推理速度，将经过 MS COCO 数据集训练的 YOLOv3 和 YOLOv3-tiny 算法模型，在 PC 端的 GTX1050Ti 显卡和国产嵌入式平台的 NNIE 硬件单元上分别进行了推理测试，测试结果如表 5.1 所示。

表 5.1　YOLOv3 和 YOLOv3-tiny 算法的推理时间对比

Table5.1　Comparison results of inference time between YOLOv3 and YOLOv3-tiny algorithm

算法模型	模型大小	测试图片尺寸	PC 推理时间	NNIE 推理时间
YOLOv3	248M	416 x 416	71ms	107ms
YOLOv3-tiny	35.4M	416 x 416	10ms	21ms

通过表格的对比结果可以看出，相比 YOLOv3，YOLOv3-tiny 在模型大小和推理速度方面更具优势，在相同大小图片的测试情况下，YOLOv3-tiny 在 NNIE 上的推理耗时大约为 YOLOv3 的五分之一；但 YOLOv3-tiny 模型在算法精度上，不如 YOLOv3 网络模型。然而，实验室无人机项目的类别检测需求相比标准的 MS COCO 数据集来说相对较少，目标无人机的型号特征也相对固定，因此，考虑到检测系统的实时性，最终选择了 YOLOv3-tiny 作为智能平台的部署模型。

5.2.1 算法原理

YOLOv3-tiny 算法的网络模型如图 5.1 所示，整体原理和 YOLOv3 算法相似，同样通过一个端到端的网络结构实现目标的检测，在 YOLOv3 原有网络结构的基础上进行了删减，原有的 Darknet-53 骨干网络被替换成了浅层的 3×3 卷积和最大池化层。

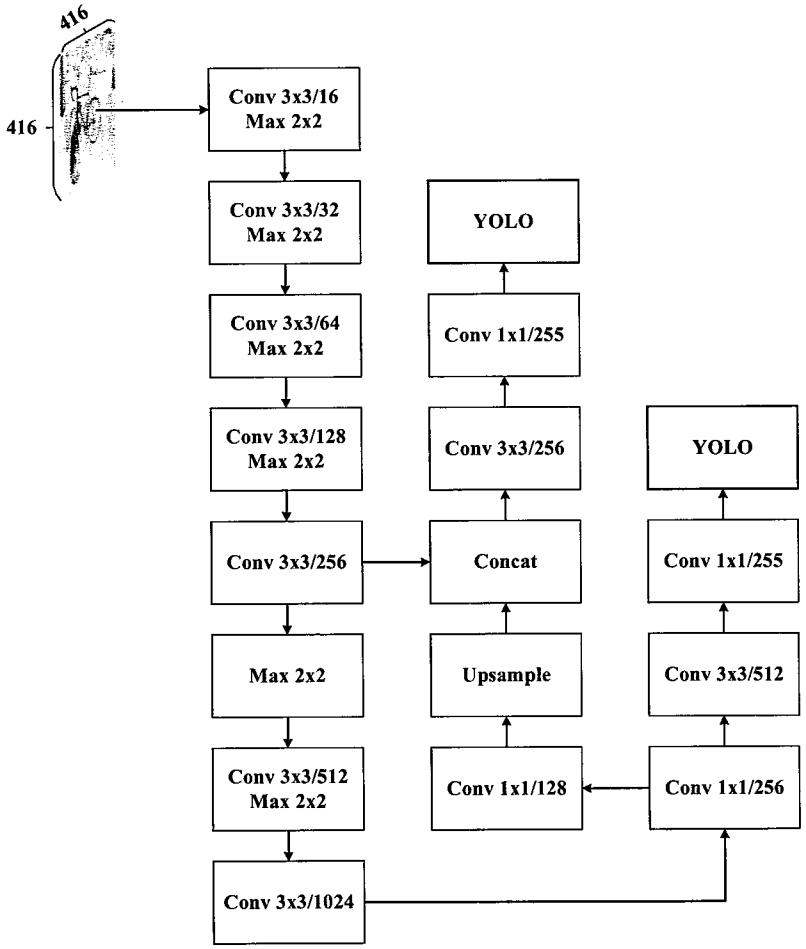


图 5.1 YOLOv3-tiny 网络结构

Figure5.1 The network structure of YOLOv3-tiny

以 416×416 大小的图像为例，YOLOv3 网络模型中 52×52、26×26 和 13×13 三种不同尺度特征的融合和预测，变为了 26×26 和 13×13 两种，原有特征融合后的多组 3×3 和 1×1 卷积操作，减少到了一组，网络深度也从原来的 106 层减少到了 23 层，网络整体更加的简洁和清晰。

5.2.2　模型优化

在实验室的无人机项目中，为了有效的训练和测试 YOLOv3-tiny 算法模型，在相关的应用场景中，采集拍摄了待检测无人机的目标图像，自建了对应的无人机图像数据库，整个数据集共有 7285 张图片，其中包括 5967 张训练图像和 1318 张测试图片，无人机类别分为固定翼（fixed）和旋翼（rotor）两种。

在原生的 YOLOv3-tiny 网络中，默认的锚点框尺寸为 COCO 数据集中聚类的结果，但本文使用的无人机数据集与开源的 COCO 数据集略有不同，因此，COCO 中聚类得到的锚点框大小，不适用于本文待检测的无人机目标。所以，本文在自建的无人机数据集上，采用 K-means 算法对无人机目标的尺寸大小进行了聚类，具体的操作步骤如下：

1. 设置 k 值 6，随机初始化 6 个锚点框样本中心；

2. 通过公式(5.1)和公式(5.2)的交并比结果，计算每个样本框到 6 个样本中心的距离，并将样本框归类到离其最近的锚点框中心 [94]；

$$d(box, centroid) = 1 - \mathrm{IOU}(box, centroid) \quad \ldots(5.1)$$

$$\mathrm{IOU} = \frac{area\left(box_{truth} \cap box_{pred}\right)}{area\left(box_{truth} \cup box_{pred}\right)} \quad \ldots(5.2)$$

3. 在第 2 步的分类完成后，从头计算聚类中心，并将得到的结果作为新的锚点框中心；

4. 重复前面的步骤 2 和步骤 3，直到最新一次的锚点框中心和上一次的聚类中心基本保持不变。

最终，在自建的无人机数据集中，得到表 5.2 所示的聚类结果。

表 5.2　聚类后的 Anchor 尺寸

Table5.2　Anchor size after clustering

Anchor	Width	Height

1	12	11
2	39	30
3	74	64
4	139	88
5	175	168
6	296	262

在得到了六种不同的锚点框后，为了进一步提高 YOLOv3-tiny 算法在智能平台 NNIE 硬件单元上的检测精度，并同时保证较快的推理速度，考虑在模型原有 416×416 的尺寸基础上，增大输入图像的尺寸提高网络特征提取的丰富度。

在 NNIE 的硬件单元中，卷积操作以 16 组数据为一个计算过程，进行并行计算，在输入通道维度为 16 倍数，输出通道和宽度的维度分别为 4 和 16 倍数的情况下，该硬件单元的计算性能和利用率可得到有效提升。因此，综合检测精度、速度和硬件计算效率，设置 512×512 为最终的输入图像尺寸。在该尺寸大小下，采用 L1 范数、activation 和梯度三种评价手段相结合的方法 [101]，对网络内卷积核的重要性进行排序，并根据其重要性的不同，剪除不同比例的尺寸大小，从而实现 YOLOv3-tiny 网络结构中部分卷积层的剪枝优化，优化结果如表 5.3 所示。

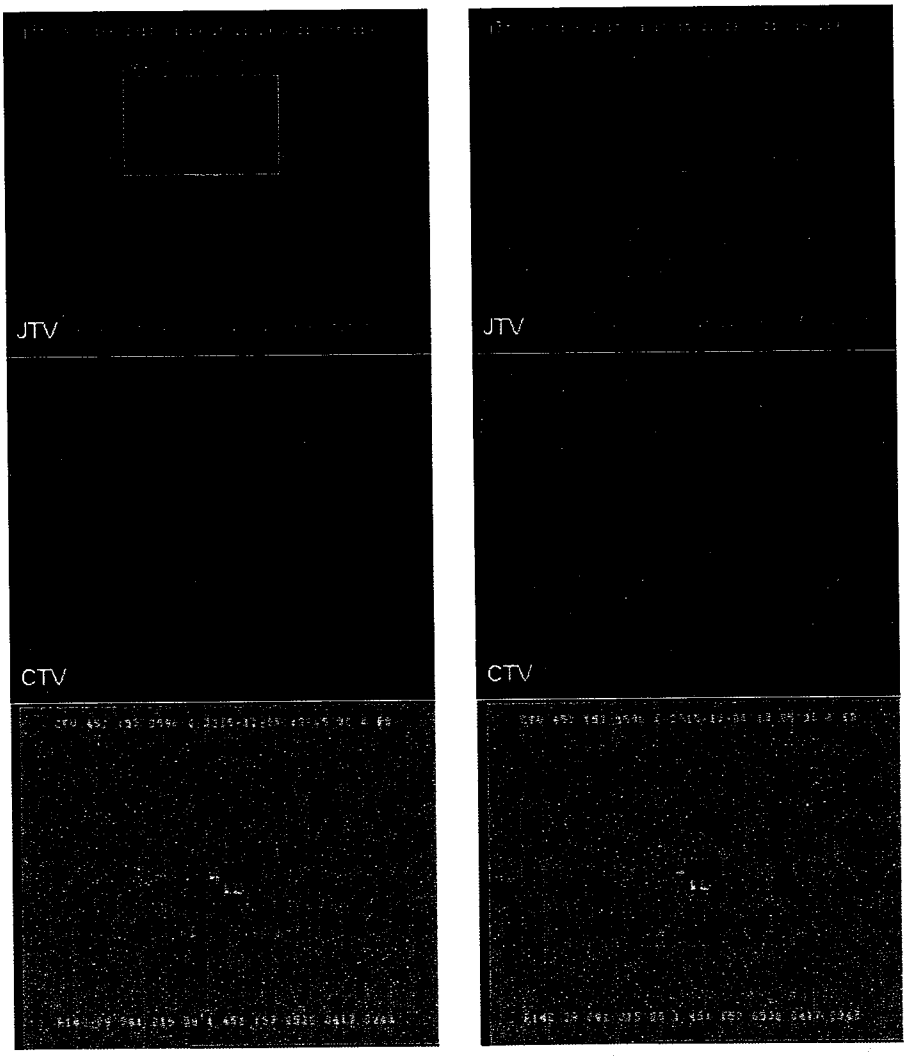
表 5.3　YOLOv3-tiny 网络结构优化结果

Table5.3　The optimization results of YOLOv3-tiny netwok structure

卷积层	优化前	BFLOPS	优化后	BFLOPS
2	3x3/32	0.604	3x3/16	0.302
4	3x3/64	0.604	3x3/32	0.151
6	3x3/128	0.604	3x3/48	0.113
8	3x3/256	0.604	3x3/96	0.085
10	3x3/512	0.604	3x3/112	0.050
12	3x3/1024	2.416	3x3/80	0.041
13	1x1/256	0.134	1x1/64	0.003
14	3x3/512	0.604	3x3/96	0.028

21	3x3/256	1.812	3x3/128	0.528

表中的第一列为卷积的层数，表示被优化卷积层在原网络结构中的第几层，第二列和第四列分别为优化前后的卷积核大小和卷积个数，第三和第五列则分为卷积优化前后的计算量。从表中可以看出，优化后的 YOLOv3-tiny 网络，对应卷积层上的计算量都产生了不同程度的下降，在 512×512 大小的输入图像下，优化后网络每层的通道和宽度输出也分别满足 NNIE 硬件单元的计算特性，能够有效的发挥 NNIE 的计算性能。



（a）　　（b）

图 5.2　YOLOv3-tiny 和优化后模型的检测对比结果

Figure5.2　YOLOv3-tiny and optimized model detection comparison results

注：(a).YOLOv3-tiny 检测结果 (b).YOLOv3-tiny 优化后的检测结果

Note：(a).YOLOv3-tiny detection results (b).YOLOv3-tiny optimized detection results

在自建的无人机数据集上，采用初始动量 0.9，权重衰减1.5×10^{-4}等训练参数，对优化后的网络结构进行了训练；整个训练在 CPU E5-2620 和 GPU GTX1080Ti 的服务器硬件环境下进行。训练完成后，在测试集上分别对 YOLOv3-tiny 和优化后的模型进行了对比测试，部分测试结果如图 5.2 所示。从检测结果可以看出，相比 YOLOv3-tiny，优化后的算法模型在无人机目标的检测过程中，大中型目标结果标注框的范围相对较大，小型目标的标注范围则相对较小；在不同的检测结果中，标注框存在某些方向延伸的现象，例如，优化后模型在第一和第三行上的检测框，分别向左和向上延伸，中间的检测结果则在上下两个方向上扩展延伸。详细的对比测试结果如表 5.4 所示，从表中可以看出，优化后的 YOLOv3-tiny 在降低了原有模型计算复杂度的同时，依然具备较好的检测能力。

表 5.4　YOLOv3-tiny 和优化后算法的精度对比

Table5.4　Comparison results of accuracy between YOLOv3-tiny and optimized network

算法模型	fixed AP(%)	rotor AP(%)
YOLOv3-tiny	89.82	89.93
优化后的 YOLOv3-tiny	87.29	88.06

表 5.5 对比了 YOLOv3-tiny 和优化后模型在 GPU GTX 1050Ti 和 NNIE 硬件单元上的推理时间。

表 5.5　YOLOv3-tiny 和优化后算法的推理时间对比

Table5.5　Comparison results of inference time between YOLOv3-tiny and optimized network

算法模型	测试图片尺寸	PC 推理时间	NNIE 推理时间
YOLOv3-tiny	512 x 512	13ms	28ms
优化后的 YOLOv3-tiny	512 x 512	8ms	10ms

在表 5.5 的时间结果中可以看出，优化后的算法在 PC 端的推理耗时上，相比优化前减少了将近 50%，在 NNIE 上则缩减到了之前的三分之一；在模型大小方面，优化后的 YOLOv3-tiny 仅有 1.8M，相比 YOLOv3-tiny 之前的 34.7M，减小了百分之九十五左右，更加有利于边缘有限资源设备的推理和存储；可见优化后的算法模型能够更好的适配国产的智能处理平台。

5.2.3　模型转换和仿真

在模型训练测试完成后，采用前文 YOLOv3 算法模型转换部分相同的方式，将优化后的 YOLOv3-tiny 模型在 Darknet 框架下的模型描述文件和权重文件，转换为 Caffe 模型下对应的*.prototxt 和*.caffemodel 文件。

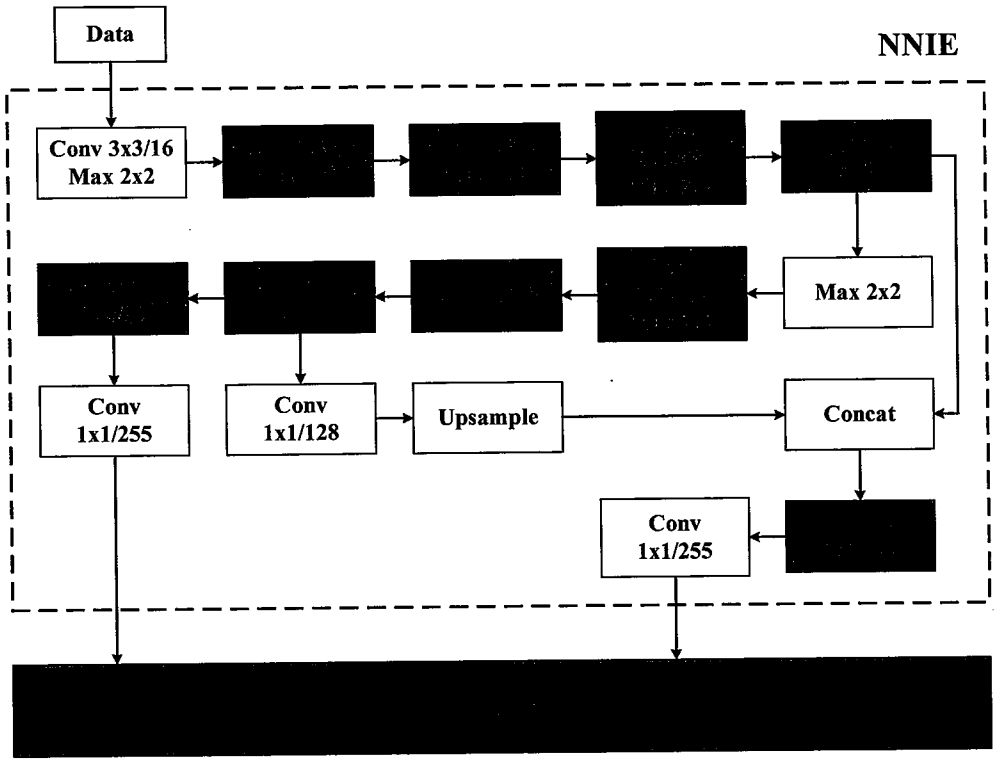


图 5.3　YOLOv3-tiny 优化网络结构的划分结果

Figure5.3　YOLOv3-tiny optimized network structure division results

在 YOLOv3-tiny 算法转换完成后，同样在 RuyiStudio 工具上创建新的 NNIE 工程，编辑 cfg 文件配置 mapper 工具的转换过程，在指定目录下生成 YOLOv3-tiny 算法模型对应的数据指令文件。在得到的指令文件中，YOLOv3-tiny 对应的优化网络结构被切分成了图 5.3 所示两部分。其中，填充蓝色的部分为优化后的

网络结构，虚线框内为 NNIE 执行的推理部分，红色矩形框中为 CPU 最后计算的 YOLO 检测层。

在 RuyiStudio 中，按照图 5.3 中 YOLOv3-tiny 网络结构的执行流程，编写对应的 C 和 C++代码，通过指令文件对应的模拟仿真库，调用智能平台的神经网络推理硬件 NNIE，对 YOLOv3-tiny 算法进行了测试，测试结果如图 5.4 所示。

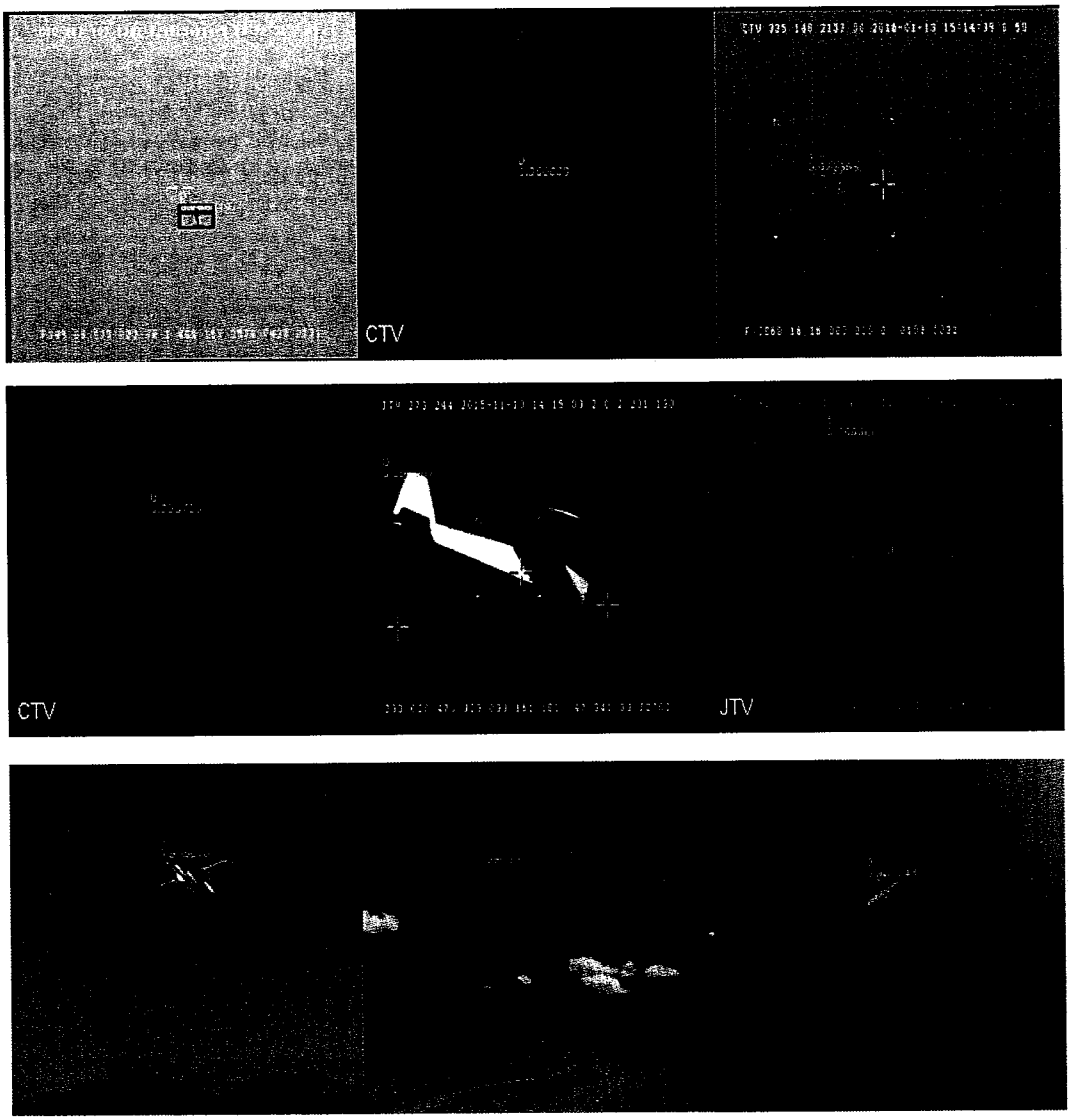


图 5.4　YOLOv3-tiny 优化后的仿真测试结果

Figure5.4　YOLOv3-tiny optimized simulation test results

在测试结果的第一和第二行中，为自建无人机数据集中的测试图片，其无人机的背景色调与训练图像中的场景具有较高的相似度，第三行为互联网中随机选取的无人机图片，包含一般应用场景下的中小尺寸无人机目标，与训练图片的差

异较大；0 代表固定翼类别，1 代表旋翼类别，从结果可以看出，在自建训练集上，第一行的小型目标和第二行的中大型目标，固定和旋翼无人机目标的检测精度都在 90%以上，在第三行差异较大的无人机场景中，固定翼和旋翼类别的检测也都取得了较好的结果，能够满足实际项目的精度需求。

5.3　平台部署实现

当 YOLOv3-tiny 算法经过调试和优化，满足无人机检测任务的实际需求后，将转换后的 YOLOv3-tiny 网络模型在国产智能平台上进行部署实现。YOLOv3-tiny 算法在平台上部署的数据流程和前文 YOLOv3 智能平台部署阶段的流程类似，都是首先通过摄像头采集图像信息，然后经由 VI 和 VPSS 模块单元分别进行处理，并在 VPSS 中划分为两个不同的通道，在通道 1 中调用 NNIE 硬件单元对采集到的数据信息进行推理；得到推理结果后，将结果传送给 VGS 单元，由该单元绘制图像中目标物体的检测框；最后，通过 VO 模块对叠加检测框后的结果进行输出显示。YOLOv3-tiny 部署实现的软件流程也与前文 YOLOv3 算法相似，同样可分为五个不同的实现阶段。接下来，对第三阶段中媒体处理平台内部视频输入、视频处理和视频输出模块的配置流程进行介绍。

在第二阶段的媒体处理平台和视频缓冲池初始配置完成后，首先对媒体处理平台内部采用并行化处理架构的 VI 模块进行配置和初始化，流程如图 5.5 所示。在 HI_MPI_SYS_Init 函数对媒体处理平台初始成功后，首先通过 HI_MPI_SYS_SetVIVPSSMode 函数，对 VI 和 VPSS 模块间的数据传输模式进行配置，从而实现 VI 预处理后的图像数据到 VPSS 模块单元的直接传送；与此同时，需要通过 MIPI 传输接口的驱动程序和底层相应的调用代码，来获取图像传感器采集到的原始数据流。在配置完成了摄像头到 VI 模块的数据通路后，调用第一阶段保存有图像传感器物理参数的全局结构体变量，按照 HI_MPI_VI_SetDevAttr 函数的使用方式，对 VI 模块中输入设备的属性信息进行配置，待配置完成后，通过调用 HI_MPI_VI_EnableDev 接口，使能 VI 模块内的输入设备；随后，通过 HI_MPI_VI_SetDevBindPipe 接口，将 VI 中的输入设备和 PIPE 管道进行绑定，对输入设备解析后的数据信息执行再处理操作，接下来按照输入设备类似的配置流程，通过 HI_MPI_VI_SetChnAttr 和 HI_MPI_VI_EnableChn 对 VI 内部的物体通道分别进行配置和使能；在这之后，

对 VI 内部的 ISP 单元进行配置，主要是将 HI_MPI_AE_Register 和 HI_MPI_AWB_Register 接口函数以回调函数的形式，把自动曝光和白平衡的库注册进 ISP 模块单元，紧接着，按照 HI_MPI_ISP_MemInit 函数的使用指南，对 ISP 所需的外部寄存器进行初始化，初始化完成后，在 HI_MPI_ISP_SetPubAttr 函数中，配置有关 ISP 的公共属性，最后通过 HI_MPI_ISP_Init 实现 ISP 模块内部 firmware 的初始化操作 [102]。

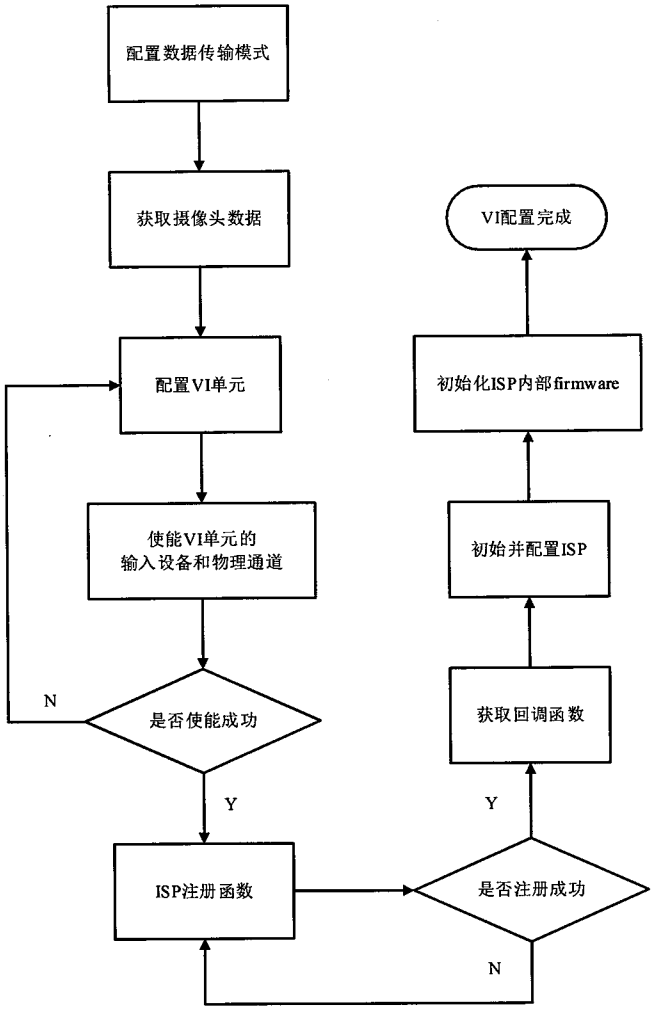


图 5.5 VI 模块配置流程

Figure5.5 The configuration process of VI module

VI 配置完成后，对 VPSS 进行设置和初始化。VPSS 在使用过程中，主要通过组和通道来实现指定的操作；其中 VPSS 以组的划分方式来分时复用 VPSS 硬

件单元，通过组内通道执行数据流的处理工作。VPSS 的整个配置流程如图 5.6 所示。

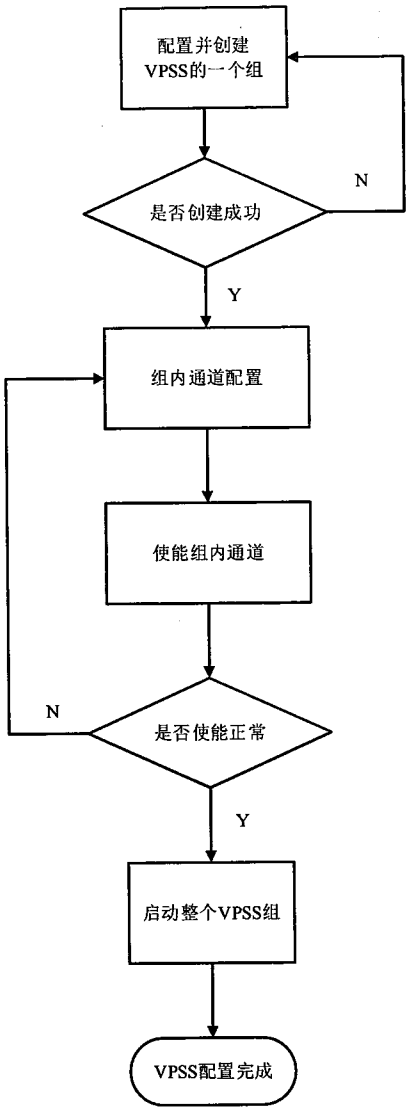


图 5.6　VPSS 模块配置流程

Figure5.6　The configuration process of VPSS module

首先根据数据的处理需求，在结构体中对 VPSS 组的属性进行配置，主要包括组内数据流格式和输入输出帧率等；配置完成后，按照函数 HI_MPI_VPSS_CreateGrp 的调用方式，传入 VPSS 属性的结构体指针，创建 VPSS 组；GROUP 创建成功后，对组内通道 0 和通道 1 的使能、图像的幅形比、动态范围 [103]和缩放等参数进行配置，配置完成后，经由 HI_MPI_VPSS_SetChnAttr 和 HI_MPI_VPSS_EnableChn 函数，分别对组内通道进行设置和启动；在组内通

道正常配置和使能的前提下，最后执行 HI_MPI_VPSS_StartGrp，启动 VPSS 的整个组[98]。

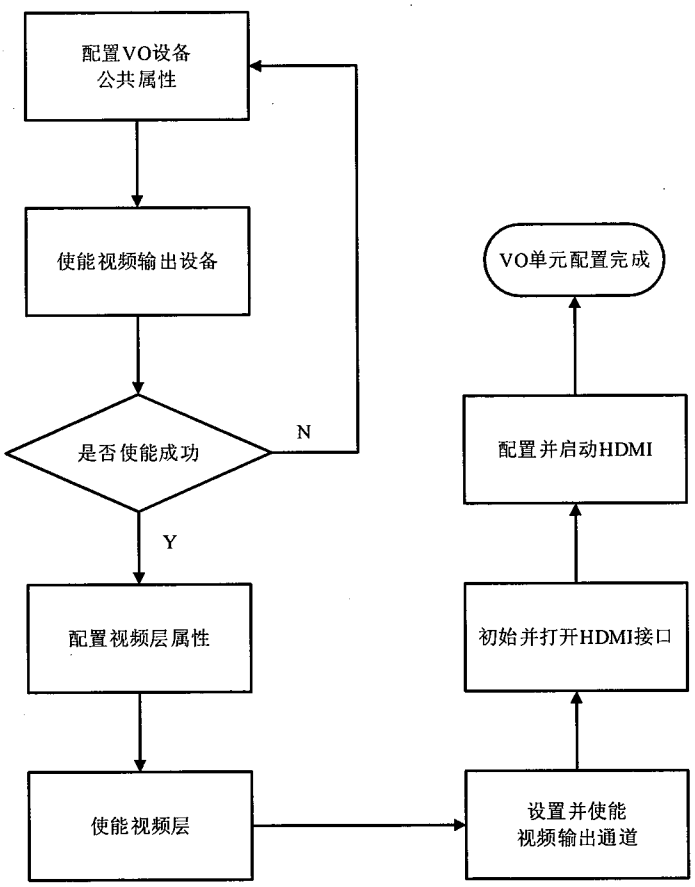


图 5.7　VO 软件配置流程

Figure5.7　VO Software configuration process

图像数据在经过 VPSS 之后的第四和第五阶段中的 NNIE、VGS 模块单元处理完成后，经由 HI_MPI_VO_SendFrame 函数传送到 VO 模块中显示输出。该模块通过内部划分的显示设备和对应设备下的视频层执行图像和视频数据输出的操作，因此，可通过配置 VO 单元中的显示设备和视频层来实现最终检测结果的输出。首先，通过调用 HI_MPI_VO_SetPubAttr，对 VO 中的输出背景色、接口类型和输出时序等属性进行配置，配置结束后，执行 HI_MPI_VO_Enable，对输出设备进行启动；启动成功后，分别通过 VO 模块对应的函数 HI_MPI_VO_SetVideoLayerAttr 和 HI_MPI_VO_EnableVideolLayer，对设备内视频层的输出动态范围和图像大小等属性进行配置和使能，视频层使能成功后，还

需要执行视频层中显示区域对应通道函数的 HI_MPI_VO_SetChnAttr 和 HI_MPI_VO_EnableChn 接口，指定视频层中通道的属性，并使能该通道。在 VO 模块内部的数据处理部分使能配置完成后，通过 HI_MPI_HDMI_Init 和 HI_MPI_HDMI_Open 函数，初始并打开平台的 HDMI 接口，接口打开成功后，通过执行 HI_MPI_HDMI_SetAttr 和 HI_MPI_HDMI_Start，对该接口进行配置和启动[98]。VO 模块整体的配置流程如前图 5.7 所示。

5.4　实现结果与分析

本文在国产的智能处理平台上，按照图 5.8 所示的方式，对优化后的 YOLOv3-tiny 网络模型进行了验证测试。图中手机实时播放待检测的无人机画面，通过较小尺寸的无人机来实时模拟无人机在现实场景下的高空飞行，而在检测平台上，通过图像传感器上安装的长焦镜头，来实现现实场景中远距离探测的能力。

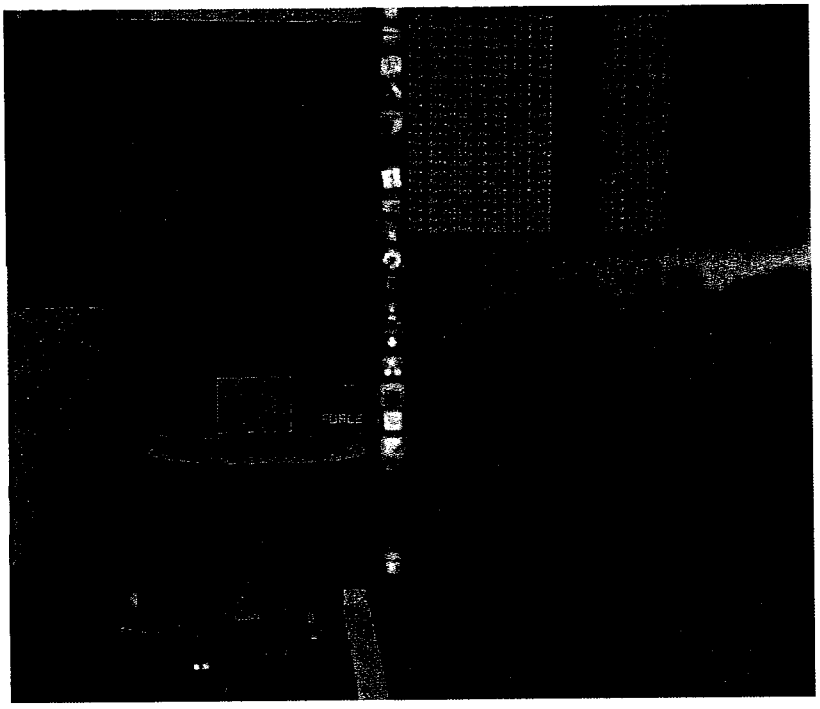


图 5.8　YOLOv3-tiny 网络的实验验证方式

Figure5.8　Experimental verification method of YOLOv3-tiny network

检测过程中，智能平台通过摄像头实时采集手机中的无人机视频画面，通过 HDMI 连接线将采集处理后的检测结果实时输出在前方显示器上，后方的显示器

则通过 Linux 服务器与目标平台实时通信，并在终端中实时打印检测的目标类别和精度。

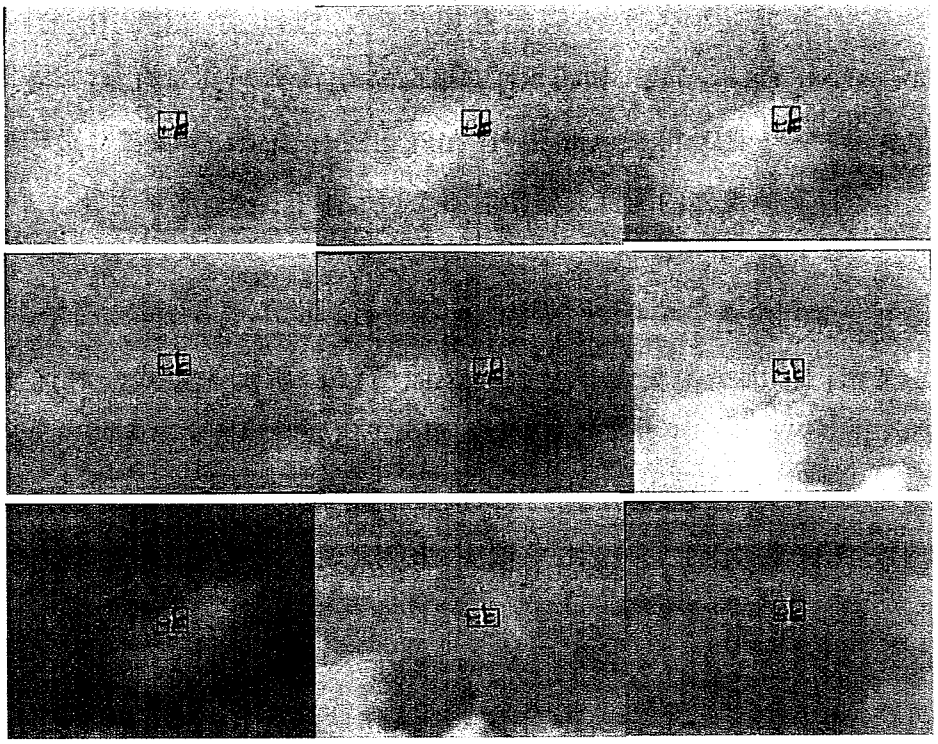


图 5.9　YOLOv3-tiny 优化后模型在智能平台上的检测结果

Figure 5.9　YOLOv3-tiny optimized model detection results on the intelligent platform

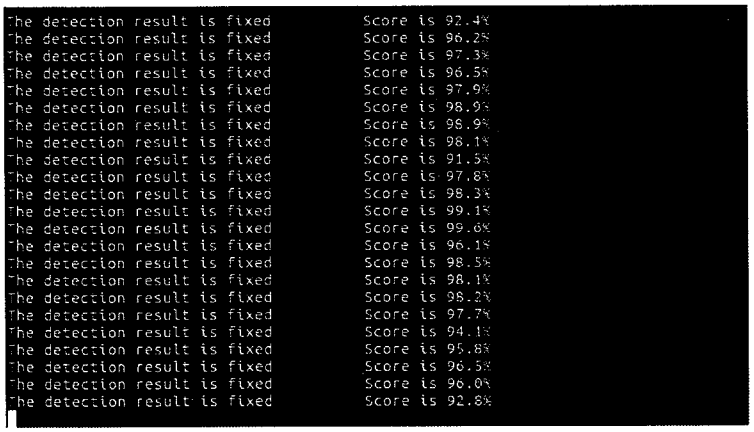


图 5.10　YOLOv3-tiny 优化模型对应的检测类别和精度

Figure5.10　Detection categories and accuracy corresponding to the YOLOv3-tiny

在图 5.8 的检测方式中，为了能够展现较为清晰的检测结果，采用视频录制的方式，对前方显示器的检测结果进行了录制，通过抽取录制结果中的部分关键

帧，获得了如图 5.9 所示的检测结果，后方显示终端上对应的检测类别和精度如图 5.10 所示。从图 5.9 和 5.10 中可以看出，在一段连续的视频序列中，基于国产智能处理平台的 YOLOv3-tiny 网络模型，能够实现稳定的检测和输出，且能在实时检测输出的过程中达到较高的识别率，可满足实际项目的检测需求。

5.5　本章小结

本章主要对国产智能平台在无人机目标检测方面的应用进行了简单介绍；首先介绍了无人机目标检测的实际应用需求，在 YOLOv3 和 YOLOv3-tiny 的网络模型中，对比选择了 YOLOv3-tiny 作为最终的部署算法；接下来，在 YOLOv3-tiny 原有的网络基础上，采用聚类的剪枝的方法，对其进行了进一步的优化，并对比了优化前后的检测结果和推理速度；之后，通过模型转换和算法仿真，对优化后的网络模型进行了验证和测试，验证测试通过后，将优化后的 YOLOv3-tiny 在国产智能平台上进行了部署实现，满足了实际项目的应用需求。

第 6 章　总结和展望

6.1　研究工作总结

目前，国外的 AI 芯片在基于深度学习的目标检测领域占据了主要的市场份额，许多典型的边缘应用场景，更是大量依赖国外芯片进行设计和研发。随着国际局势的复杂多变，国外先进技术对我国的逐步封锁和关键领域的“卡脖子”，基于国外 AI 芯片开发的目标检测应用，在稳定性、安全性和可持续性等方面都存在不可预知的隐患。因此，基于国产 AI 芯片的目标检测算法研究，意义重大，符合国家未来的发展需求。

1. 本文从 AI 芯片的性能、功耗、扩展性和可持续发展等方面，对国内的同类芯片进行了对比和分析，最终选用了海思的 Hi3559AV100 芯片，并针对该国产芯片开发流程复杂和开发难度大的问题，配置和搭建了统一高效的集成开发环境，大大简化了整体的开发流程和难度；在基于该国产芯片处理平台的有限资源下，针对深度学习的目标检测算法在边缘端的应用部署需求，对智能平台的整体架构进行了研究设计与部署，并在整体的数据处理流程下，通过硬件单元的配置管理和软件系统的适配、编译与烧录，最大化的利用了平台的资源，形成了一套通用的基于该国产 AI 芯片的拥有自主知识产权、安全可靠和精简高效的智能处理平台。

2. 在基于海思的智能处理平台上，开展了目标检测算法的部署研究。根据目标检测算法的特点与算法部署实现的需求，在搭建的集成开发环境基础上，研究实现了两阶段算法模型 R-FCN 与单阶段算法模型 YOLOv3 的网络模型转换、算法仿真验证和智能平台有限资源下的协同部署流程，并在设计的单芯片并行处理架构下，完成了两种网络模型的检测推理，在最终的结果输出中，提高了百分之二十一左右的检测帧率；最后，通过公开的 changedetection 数据集，对部署后的两种算法进行了对比测试，结果显示，YOLOv3 在对连续视频目标进行检测的过程中，检测稳定度要优于采用 ResNet-50 的 R-FCN 算法，在 NNIE 的推理速度方面，由于 YOLOv3 的 Darknet-53 对比 ResNet-50 骨干网络的相对复杂，R-FCN 则在推理耗时上更具优势。

3. 根据无人机目标检测的实际应用需求，对比选取了 YOLOv3-tiny 网络模型，并在自建的无人机数据集和 Hi3559AV100 中神经网络推理的 NNIE 硬件单元下，通过聚类算法和集成剪枝的方法，对 YOLOv3-tiny 网络的锚点框和网络结构进行了优化；在保证了网络检测精度的同时，缩减了百分之九十五左右的模型大小，并将模型的推理速度提高到了原来的三倍左右；之后，通过模型转换和算法仿真，对网络模型进行了验证和测试，并将优化后的模型在国产智能平台上进行了部署，实现了精度百分之九十以上的检测输出，满足了项目的实际需求，验证了智能处理平台在实际应用中的可靠性和有效性。最终，形成了智能处理平台与算法部署的整体方案，在国产的 AI 芯片上开发出了稳定可靠的智能检测平台。

6.2 未来展望

基于国产 AI 芯片开发的智能处理平台，在功能和性能上能够满足大部分的应用需求。但是在未来智能应用的不断发展过程中，还存在一定的优化空间，进一步的研究可以从以下几个方面展开：

1. 可根据智能平台中神经网络推理硬件 NNIE 的硬件架构和特性，针对不同应用场景和需求，有针对性的设计专用的网络结构，更大程度上的提高该计算单元的计算效率和性能；

2. 在模型算法的部署实现过程中，可通过编写多线程的代码程序，提高 CPU 硬件对网络模型中非 NNIE 执行部分和推理结果后处理部分的计算效率，提升系统平台整体的利用率；

3. 在现有的检测平台上，通过增加可视化的控制界面，对智能平台直接进行控制，可省去第三方设备接入控制的复杂过程；除此之外，还可以按照应用场景的实际需求，在智能平台上添加专用的显示单元，直接在边缘端进行实时显示，从而实现整个平台采集、处理、显示和控制的一体化。