2022-2023 学年冬季学期

《研究方法与前沿(计算机)》(0830SY02)

课程报告

成绩 (百分制)	

学　　号	20121802	学　　院	计算机工程与科学学院
姓　　名	严昕宇	手工签名	
报告题目	基于寒武纪 MLU 与 EAST 模型的文本检测应用研究		

教师评语

教师签名

批阅日期　　年　月　日

上海大学 计算机工程与科学学院

基于寒武纪 MLU 与 EAST 模型的文本识别应用研究

严昕宇 20121802

(上海大学 计算机工程与科学学院)

摘 要 寒武纪思元 270 是一款新型 DLP 深度学习处理器．与 GPU 相比，其专门面向深度学习计算，设计了专用计算单元，且基于寒武纪的软件栈与 GPU 有所差异。现有研究大多专注于 GPU 上的深度学习负载性能分析和优化，由于寒武纪平台推出不久且具有新的体系结构特征，其实际表现仍有待探索。为深入挖掘寒武纪的性能和优化方法，本文依据现有的文本检测的深度学习模型 EAST，提出基于搭载有寒武纪思元 270 DLP 芯片的深度学习加速卡 MLU270 加速模型计算速度,并将用 BCL 实现的 Split+Sub+Concat 合并算子集成到 TensorFlow 框架中的优化手段，通过减少数据搬运以实现更高效的识别。

关键词 深度学习；EAST；寒武纪；深度学习处理器；文本识别

1 引言

1.1 研究背景及意义

随着科学技术的快速发展，手机、电脑等电子产品已经深入每个人的日常生活之中，人们因而可以更加便捷地从外界获得自己所要的信息。且在这个信息化的时代，由于多媒体技术的蓬勃发展，人类的信息交流中图像所占的比重越来越大。图像中包含了非常丰富的信息，但是长久以来，只有人能够从图像中获取有用的信息，即使能够做非常复杂的逻辑计算的机器，往往对最简单的图像也无法识别。近些年来，随着人工智能技术的发展并逐渐融入到生活中的各个角落，各种场景对图像内容的理解提出了越来越高的要求，所以利用计算机视觉的技术应运而生。

通常来说，计算机视觉是利用计算机及相关设备对生物视觉进行模拟，从而实现对获取的信息进行自动分析处理，提取出人类说需要的部分，如车牌识别、人脸识别、自动驾驶、医学图像分析等方面。所以如何更加有效的从视频、图像中获得信息，是现在重要的研究方向。

从传说中的仓颉造字以来，文字或文本就为人类的发展的进步带来了重要贡献。到现在为止，文本依旧是包含丰富而准确信息的一个大类场景。图像中出现的文本作为图像中的一个重要内容，往往对视觉信息的表达有着关键作用。因此，自然场景中的文本检测已成为计算机视觉和文档分析中重要且活跃的研究主题，尤其是近年来，尽管仍然存在各种挑战(例如噪声，模糊，失真，遮挡和变化)，但在这些领域的研究工作仍取得了实质性、突破性进展。

通过对图像中的文本进行检测识别，极大地方便人类从各种反面获取文本知识。在实际应用中，例如现在的盲人可以利用识别图像文本的机器进行阅读书本，帮助残疾人士学习；帮助车辆识别路面交通的文字，识别车牌号码，提升道路安全；通过对身份证的检测识别，不仅加快业务的处理速度，而且减少人力物力的消耗等等。在新闻出版，邮政快递，金融服务等其他领域都广泛使用了文本识别算法[1]。

例如，对于如图 1.1 所示的针对会计领域的发票文本识别，就大大减轻了人工比对的负担。

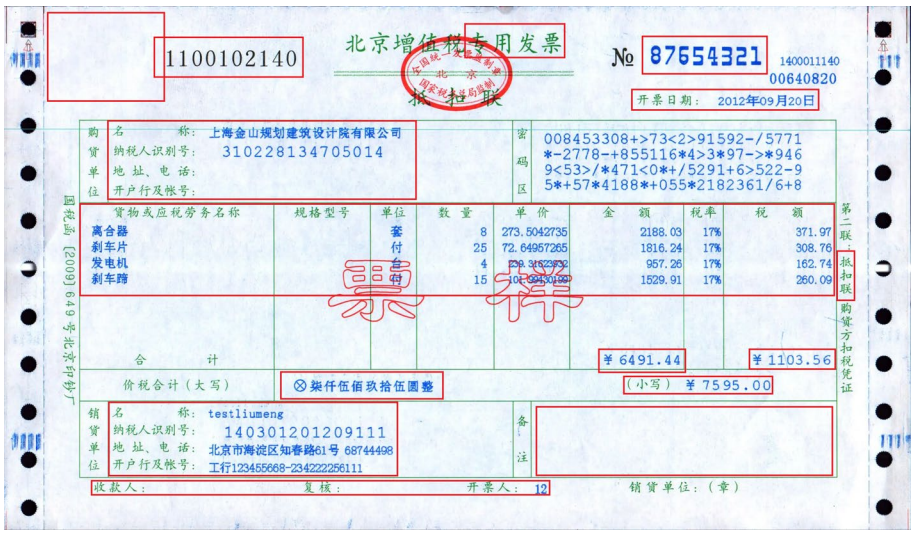


图 1.1 OCR 应用举例

光学字符识别(Optical Character Recognition, OCR)是指对文本资料的图像文本进行文字识别，获取文本资料版面信息的过程。OCR 算法的研究有很长的历史，现在对于传统光学字符识别的文本检测与识别技术已经相当成熟，因为其背景简单干净，字体工整，识别率很高。

OCR 大致流程如下所示：

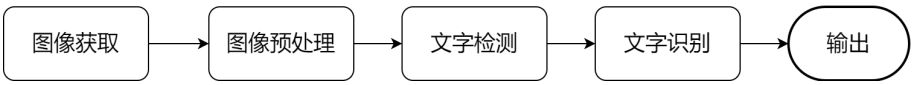


图 1.2 OCR 流程

然而，自然场景的图像非常复杂，沿用印刷体的检测技术会对文本识别的精确性和准确率造成很大的影响。而且在对文本的研究中，由于文本位置的不确定性，图像采集的时受外界的干扰，都会对文本检测带来巨大的困难，给在自然场景下的文本检测带来了诸多挑战，其中主要包括：文本语言种类的多样性、文本组成的多样性、图像背景的复杂性以及图像的噪声等。

综上所述，由于自然场景具有随机的特点，文本检测上不仅对于理论有较高的研究价值，而且在实际应用中也扮演着重要角色。因此在近些年，基于各种深度学习的各种算法逐渐成为重要的研究方向，即从一开始人工文本特征的过滤筛选，到现在由于神经网络的迅猛发展，用神经网络来处理特征加快了检测准确度。但是由于数据的增加，这些训练模型往往非常庞大，这就对检测效率要求更高，也使得文本检测有较大的上升发展空间。国内外众多研究者都致力于是的文本检测发挥更大的应用价值。本论文为了提高文本检测模型的运行效率，就此展开研究。

1.2 国内外研究现状

在计算机视觉中文本检测是一个重要的组成部分，较为早期的时候就已经出现了。在 20 世纪 90 年代中期 Ohya J、Zhong Y、Lee CM 等率先开展了对自然场景下文本检测研究。之后经过几十年的年展，并且在深度学习出现的后，大量检测方法涌现出来，文本检测领域出现很多行之有效的检测方法。最近，在目标检测技术与语义分割技术方面更是取得了重要进步，在文本检测方面也有这很大的突破，检测的准确率和鲁棒性也更高[2]。

就文本检测方法的发展来看，自然场景下文本检测方法大体上可以分为以下两类：1.传统的自然场景文本检测方法；2.基于深度学习的自然场景文本检测方法。

1.2.1 传统的自然场景文本检测方法

在传统的自然场景检测方法中，主要工作就是进行手工特征的设计，获得候选文本区域，接着对这候选区域进行验证，过滤出文本区域。而对于传统的自然场景文本检测算法，根据检测的方式不同，又能分为以下两种方法：基于滑动窗口的文本检测和基于连通域的文本检测[3]。

上述中的传统的文本检测算法都是使用人工设计特征，利用分类器进行过滤。所以设计一个合理有效的人工特征，对于检测的效果来说非常重要，在复杂的背景下难以具有鲁棒性，而且在使用大量复杂的人工特征后，还会增加计算的复杂度，让检测效率不理想，所以现在经典的文本检测算法满足不了现代人对检测结果的迅速和准确的要求。因此，开始将深度学习的概念引入文本检测领域后，提出使用深度学习来解决文本检测中的对于复杂图片处理的鲁棒性和准确性问题。

1.2.2 基于深度学习的自然场景文本检测方法

由于手工设计的特征分类能力的不足，文本检测性能在较长的一段时间内难以取得较大突破，直至出现深度学习技术，人们对于深度学习(Deep Learning, DL)开始关注。而其中的一些算法，像卷积神经网络CNN(Convolution Neural Network)以及递归神经网络 RNN(Recurrent Neural Network)等作为代表算法，经过一系列的发展，对图像处理和语音上的处理上有着很好的效果。

与传统的自然场景下文本检测算法相比，利用深度学习的文本检测算法的优点更加突出，它能避免设计繁杂的手工特征提取分类器，提升检测效率，并且通过图像深层的整体特征与浅层外观特征来表示属性类别，使计算机能够清楚认识文本的有效特征，便于文本检测。目前已出现大量的基于深度学习的文本检测算法，其主要包括以下几种深度学习算法：R-CNN、Faster-R-CNN、YOLO、SSD 等算法[4]。

1.3 本文研究内容

近年来，深度学习取得突破性进展很大程度上得益于芯片和算力的支撑。英伟达的图形处理器(Graphics Processing Unit, GPU)凭借其高性能和通用性的特点，一直在深度学习训练任务中占据主要地位，主流的深度学习工具如 TensorFlow、PyTorch 等都会基于英伟达 GPU 产品进行研发。为了获取更高的性能，学术界和工业界开始将研究重点从通用计算架构(General Purpose Architecture)转向领域专用架构(Domain Specific Architecture, DSA)。各厂商正积极研发神经网络加速器(Neural Network Accelerator)，以及基于神经网络加速器的编译器、库、优化工具和上层软件：Google 推出了张量处理器(Tensor Processing Unit, TPU)并将其大量部署在云数据中心，Google 开源的深度学习框架 TensorFlow 可以充分利用 TPU 的算力；MIT 提出了一种高能效、可重配置的神经网络加速器 Eyeriss，可以加速卷积神经网络的训练；华为发布了异腾(Ascend)系列处理器，包括面向训练场景的异腾 910 和面向推理场景的异腾 310；寒武纪基于 DianNao 系列指令集，并推出思源系列 DLP 芯片及深度学习加速卡 MLU，这也是本文中所使用的硬件平台。

本文依据现有的文本检测的深度学习模型 EAST，提出基于搭载有寒武纪 AI 芯片的寒武纪深度学习加速卡 MLU 加速模型计算速度，并将用 BCL 实现的 Split+Sub+Concat 合并算子集成到 TensorFlow 框架中的优化手段，通过减少数据搬运以实现更高效的识别。

1.4 本文章节安排

本文的第 2 节将简述人工智能芯片的定义、发展历程和系统架构；第 3 节将介绍深度学习的训练过程；第 4 节则是介绍编译软件栈与深度学习框架；第 5 节对 EAST 模型进行分析；第 6 节介绍实验环境及实验内容；第 7 节介绍实验步骤及实验结果；第 8 节对全文进行总结。

2 人工智能芯片

当今世界，人工智能正被广泛运用于各式各样的场景中。在 2020 年世界人工智能大会云端峰会上，会上，中国科学院院士、清华大学人工智能研究院院长张钹教授提出了人工智能“四要素”：知识、数据、算法与算力。其中，算力正是依靠运行 AI 算法的芯片和与之相对应组成的计算平台。与此同时，伴随着人们日益增长的需求，以及增长的所需处理数据量，AI 算法必须能够高效的运行在硬件平台上。人工智能芯片，即 AI 芯片正是解决这个问题的关键。

广义上讲，只要能够运行人工智能算法的芯片都叫做 AI 芯片。但是通常意义上的 AI 芯片，指的是针对 AI 算法做了特殊加速设计的芯片，当前阶段的 AI 算法以深度学习算法为主，也包括其它机器学习算法。

AI 芯片的发展，经历了从 CPU 到 GPU，再到 FPGA、ASIC 与类脑芯片的发展历程。

2.1发展概述

在 20 世纪 90 年代和 21 世纪初，受制于算法、数据量等因素制约，且当初人工智能研究流派(即符号主义)的影响，这个阶段 AI 芯片并没有太多市场需求，通用 CPU 即可满足 AI 算法的应用需求。

从 2006 年开始，Geoffrey Hinton 在 Science 杂志上发表了使用神经网络降低数据维度的文章，为深度学习带来研究理论基础。同年，英伟达推出了 CUDA，使开发者可以在 GPU 上面方便地进行编程。在这之后的 5 年，人们发现 GPU 并行计算能力恰好适应 AI 算法和数据并行计算的需求，开始逐渐地尝试使用 GPU 来运行 AI 算法和进行验证。

2012 年后，云计算和移动设备广泛的发展，使得大量图像数据和用户数据增加。同年 Google 使用 1.6 万个 GPU 核组成的并行计算平台 Google Brain 训练 AI 模型，在语音和图像识别等领域获得了巨大成功。同年 Alex 利用 2 块 GPU 的并行能力，实现的 AlexNet，将图像分类错误率从 26%降低到 15%，以碾压第二名的分类性能，一举夺得 ImageNet 比赛冠军。

而产业和学术界对算力的需求是永无止尽的，随着 AI 对计算的需求不断增长和 AI 产业的爆发，2014 年英伟达发布了首个为深度学习设计的 GPU 架构 Pascal。2016 年，Google 公布了第一代 ASIC 芯片 TPU，到 2021 年已发展到第四代的 TPU v4。

在国内，2016 年寒武纪发布了用于云计算和推理的 DianNao FPGA。2017 年，华为第一个将 NPU 集成到手机上，使用 AI 算法增强手机拍照能力。在 2018 年，华为发布昇腾系列芯片 310 和 910，其中 910 芯片实现了 7nm EUV 工艺、32 核达芬奇架构，FP16 算力达到 256 Tera-FLOPS。2021 年 1 月，寒武纪首颗 7nm 训练芯片思元 290 正式亮相，是寒武纪的首颗训练芯片，采用 7nm 工艺，集成 460 亿个晶体管，支持 MLUv02 扩展架构，全面支持 AI 训练、推理或混合型人工智能计算加速任务。

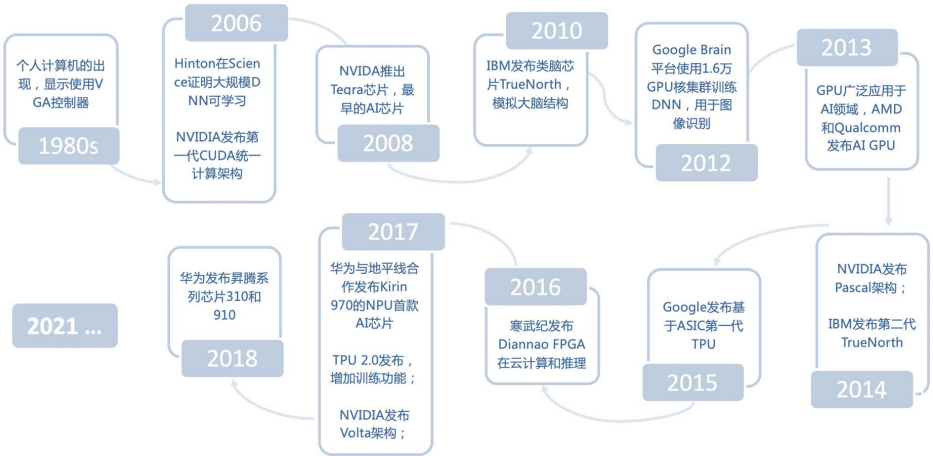


图 2.1 AI 芯片的发展过程

2.2 系统架构

在传统 CPU 结构中，除了数据运算，还需要执行数据的存储与读取、指令分析、分支跳转等命令，因此其擅长的是逻辑控制、串行运算。但 AI 算法通常需要对海量数据进行处理，使用 CPU 执行算法，将会花费大量的时间在数据指令的读取分析上，因此计算效率非常低[5]。

随着 AI 产业的发展，业界出现了 4 种 AI 芯片架构。以冯·诺依曼传统计算架构为基础，用于加速硬件计算能力为主，有 GPU、FPGA、ASIC 这 3 种类型为代表；另外是颠覆冯·诺依曼架构，采用类脑神经结构独立设计，来提升计算能力的类脑芯片。下面将详细展开，分析 4 种不同类型的架构。

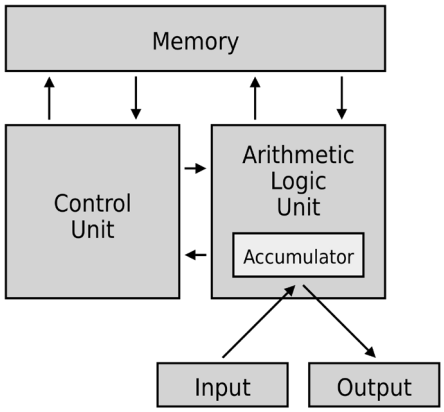
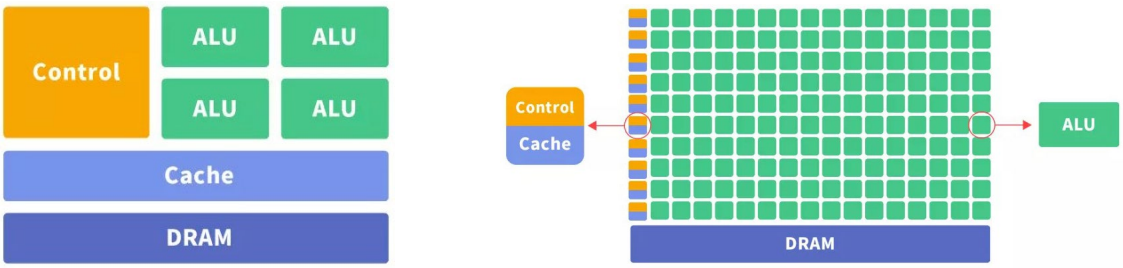


图 2.2 冯·诺依曼架构

2.2.1 图形处理器 GPU

GPU(Graphics Processing Unit)，即图形处理器，是一种由大量运算单元组成的大规模并行计算架构，早先由 CPU 中分出来专门用于处理图像并行计算数据，专为同时处理多重并行计算任务而设计。

GPU 主要面对类型高度统一、相互无依赖的大规模数据和不需打断的纯净计算环境。GPU 中也包含基本的计算单元、控制单元和存储单元，但 GPU 的架构与 CPU 有很大不同，其架构图如下所示。



(a) CPU **(b) GPU**

图 2.3 CPU 与 GPU 架构对比

可以发现，CPU 芯片空间的不到 20%是 ALU，与 CPU 相比，GPU 拥有若干由数以千计的更小的核心(专为同时处理多重任务而设计)组成的大规模并行计算架构，芯片空间的 80%以上是 ALU。即 GPU 基于大吞吐量设计，拥有更多的 ALU 用于数据并行处理。且 GPU 内部大部分的晶体管可以组成各类专用电路、多条流水线，使得 GPU 的计算速度远高于 CPU，更擅长计算密集且易于并行的程序，拥有更加强大的浮点运算能力。

正是由于 GPU 在矩阵计算和并行计算上具有突出的性能，且作为异构计算的主力，最早作为深度学习的加速芯片被引入 AI 领域，生态成熟，因此广泛使用在深度学习算法领域。

作为 GPU 领域内的巨头，NVIDIA 沿用 GPU 架构，对深度学习主要向两个方向发力：

- **丰富生态**：推出 cuDNN 针对神经网络的优化库，提升易用性并优化 GPU 底层架构
- **提升定制性**：增加多数据类型支持(不再坚持 float32，增加 int8 等)；添加深度学习专用模块(如引入并配备张量核的改进型架构，V100 的 TensorCore)

当然 GPU 也有其自身的不足，如成本高、能耗比低、延迟高等问题。

2.2.2 现场可编程逻辑阵列 FPGA

FPGA (Field Programmable Gate Array)，即现场可编程逻辑阵列，是一种集成大量基本门电路及存储器的芯片，通过烧入 FPGA 配置文件定义门电路及存储器间的连线，实现特定的功能，用户可根据自身需要可重复编程的集成电路。通俗理解就是，可以把硬件设计重复烧写在它的可编程存储器里面，使 FPGA 芯片可以执行不同的硬件设计和功能，所以叫做现场可编程逻辑阵列，如下图所示。

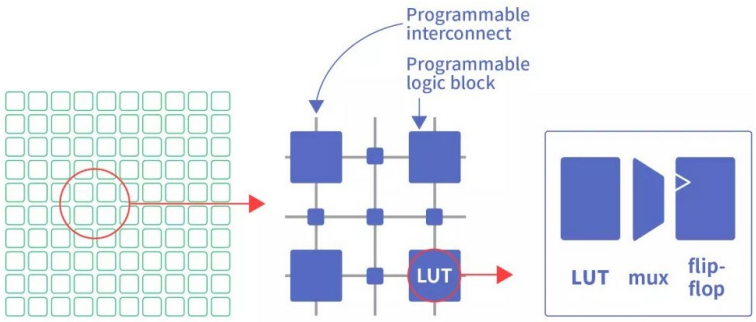


图 2.4 FPGA 内部架构图

FPGA 将指令锁定在硬件架构上，然后使用硬件指令流运行数据，即将 AI 的计算架构用硬件电路实现出来，然后持续的将数据流输入系统，并完成计算。

与 GPU 不同的是，FPGA 可以同时拥有硬件流水线并行和数据并行处理能力，适用于以硬件流水线方式处理数据流，适合处理小计算量大批次的计算任务，对于大量矩阵运算具有低延迟的特点，适合在推断环节支撑海量的用户实时计算请求。因此，FPGA 非常适用于 AI 推理阶段，相对于 CPU 与 GPU 有明显的性能或者能耗优势。FPGA 可灵活支持各类深度学习的计算任务，目前已被大量应用于深度神经网络模型的加速器。

2.2.3 专用集成电路 ASIC

由于 FPGA 编程难度大，对开发者要求高，于是出现了 ASIC。ASIC(Application Specific Integrated Circuit)，即专用集成电路，是一种为专门目的而设计的集成电路。近些年来涌现的例如 TPU、NPU、VPU、BPU 等各种 XPU 芯片，本质上都属于 ASIC。

在集成电路界 ASIC 被认为是一种为专门目的而设计的集成电路。它是指应特定用户要求和特定电子系统的需要而设计、制造的集成电路。ASIC 的特点是面向特定用户的需求，基于专门为特定 AI 算法设计出来的架构。因此 ASIC 在批量生产时与通用集成电路相比，具有体积更小、功耗更低、可靠性提高、性能提高、保密性增强、成本降低等优点。缺点是电路设计定制，导致相对开发周期长，不易修改和扩展。

目前，大多是具备 AI 算法又擅长芯片研发的巨头参与，如 Google 的 TPU、华为昇腾系列芯片、寒武纪、比特大陆、地平线等。由于完美适用于神经网络相关算法，ASIC 在性能和功耗上都要优于 GPU 和 FPGA，TPU v1 是传统 GPU 性能的 14-16 倍。因此，大部分学者预计，ASIC 将是未来 AI 芯片的核心。

2.2.4 类脑芯片

有一种观点，即真正的人工智能芯片，未来发展的方向是类脑芯片。类脑芯片直接基于神经形态架构设计，用于模拟人脑功能进行感知方式、行为方式和思维方式的计算。但是研发难度巨大。类脑芯片不采用经典的冯诺依曼架构，而是基于神经形态架构设计，以 TrueNorth 和清华大学天机芯片为代表。

IBM 研究人员将存储单元作为突触、计算单元作为神经元、传输单元作为轴突搭建了神经芯片的原型。2011 年 8 月，IBM 率先在类脑芯片上取得进展，他们在模拟人脑大脑结构基础上，研发出两个具有感知、认知功能的硅芯片原型。但因技术上的限制，IBM 戏称第一代 TrueNorth 为“虫脑”。2014 年 TrueNorth 第二代诞生，其性能相比于第一代有了较大提升。IBM 称如果 48 颗 TrueNorth 芯片组建起具有 4800 万个神经元的网络，那这 48 颗芯片带来的智力水平将相似于普通老鼠。但从 2014 年亮相后，这款芯片一直没用大的进展。

图 2.5 IBM TrueNorth 芯片

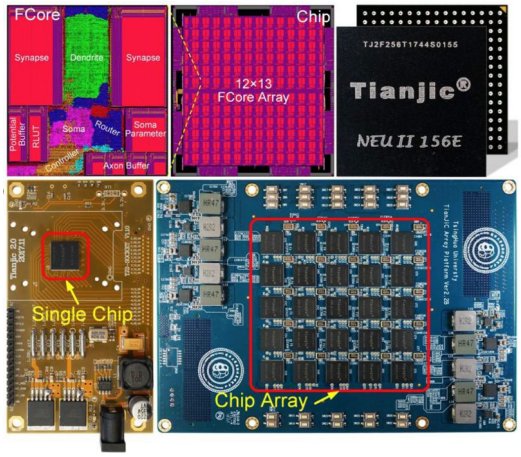


图 2.6 清华大学天机芯片

清华大学类脑计算中心于 2015 年 11 月成功的研制了国内首款超大规模的神经形态类脑计算天机芯片。“第一代芯片的制程约为 110 纳米，只是个 DEMO。2017 年，团队研发了第二代“天机芯”芯片。第二代“天机芯”具有高速度、高性能、低功耗的特点，制程缩小至 28 纳米。相比于当前世界先进的 IBM 的 TrueNorth 芯片，其功能更全、灵活性和扩展性更好。2019 年 8 月，第二代“天机芯”(Tianjic)登上了《自然》(Nature)封面，作为人工通用智能领域的一个重磅应用案例进行了展示。

2.2.5 总结及对比

表 2.1 人工智能芯片的系统架构对比

类别	CPU	GPU	FPGA	ASIC	类脑芯片
定制化程度	通用性	通用性	半定制性	定制性	半定制性
功耗	中	高	低	低	低
成本	高	高	中	低	高
算力	低	中	高	高	中
代表公司	Intel AMD	AMD NVIDIA	Intel Xilinx 深鉴科技	Google 华为 寒武纪 地平线	IBM Intel Qualcomm 西井科技

2.3 神经网络处理器NPU

NPU(Neural network Processing Unit)，即神经网络处理器。NPU 作为一种主要采用 ASIC 技术的专用嵌入式神经网络芯片，使用 DSA(Domain Specific Architecture)克服了 CPU、GPU 等通用处理器在深度学习等领域数据吞吐量、算力的限制，大幅提高端侧、嵌入式设备的处理性能。现今主要的 NPU 集中在推理芯片领域，使用 NPU 等技术的异构计算处理器使得图像数据的端侧处理、加强，主体追踪成为可能，也使得传统手机应用、嵌入式机器人领域、自动驾驶等走向大众化。NPU 的出现代表芯片从通用化逐渐开始走向领域专用芯片的异构混合计算。

2.3.1 华为异腾

异腾系列加速器是华为公司推出的一款面向神经网络加速的 DSA 硬件，被称为 Neural Network Processing Unit(NPU)。异腾加速器是一个片上系统(System on Chip)，共有两类 AI 计算引擎：AICore 和 AI CPU，其中 AI Core 提供神经网络算力，AI CPU 用于承担非矩阵类计算。

图 1 为 AI Core 所基于的华为自研的 DaVinci 架构。AI Core 有三类计算单元，三类计算单元分别支持不同类型的深度学习计算任务：专门负责乘累加的矩阵计算单元(Cube Unit)，负责激活函数等非乘累加计算的向量计算单元(Vector Unit)和高度灵活的标量计算单元(Scalar Unit)。矩阵计算单元使用 FP16 精度，算力强大，但只能完成乘累加计算。相比矩阵计算单元，向量计算单元的指令更加丰富，可以完成激活函数等非乘累加类计算。标量计算单元主要完成循环控制、分支判断等标量计算[6]。

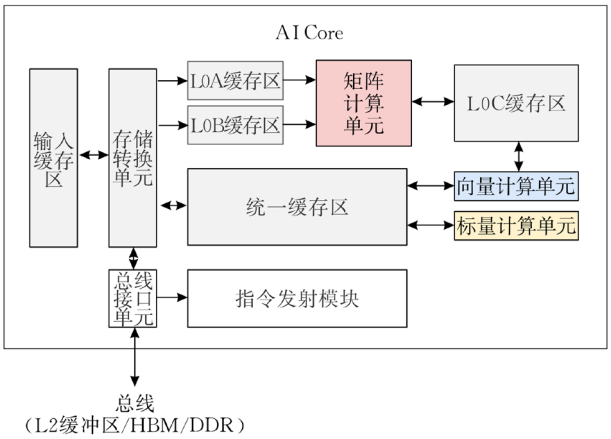


图 2.7 AI Core 架构示意图

2.3.2 寒武纪

寒武纪科技是全球第一个成功流片并拥有成熟产品的人工智能芯片公司，目前主要拥有三种处理器结构，即寒武纪 1 号(DianNao，面向神经网络的原型处理器结构)、寒武纪 2 号(DaDianNao，面向大规模神经网络)和寒武纪 3 号(PuDianNao，面向多种机器学习算法)。

2017 年，华为麒麟 970 处理器中搭载的 NPU 即寒武纪 1 号的衍生型号 Cambricon-1A 处理器，并连续迭代推出 1H、1M 系列芯片。同年，寒武纪公布机器学习处理器 MLU(属于 ASIC)的研发规划，以布局云端训练环节的和深度学习处理器(DLP, Deep Learning Processor)思元 100 和思元 270。又于 2019 年推出基于思元 220 芯片的边缘智能加速卡。2021 年初推出的云端训练新品思元 290，在性能功耗比上已与英伟达 A100、昇腾 910 对齐。2021 年 11 月，推出的云端推训一体芯片思元 370，采用 Chiplet 技术，7nm 制程，最大算力 256 TOPS。

3 深度学习训练

3.1 模型泛化能力

深度学习的主要目的是在已有训练数据集上训练模型，使用该模型在未知数据上进行高质量的预测，即模型必须有泛化能力。为了量化模型在未知数据上的预测能力，模型训练时通常需要使用一个验证数据集来验证模型的预测能力，验证数据集通常区别于训练数据集。当模型在验证集上的效果表现良好时，会停止迭代。

3.2 典型训练过程

训练数据集的特征经过神经网络后，使用损失函数与标签比对，得到损失值；训练时需要选择一种最优化算法，通过多轮迭代，优化损失值。训练过程中的最优化算法一般基于随机梯度下降(Stochastic Gradient Descent, SGD)或其变体，如 Adam。限于硬件内存大小，优化算法每次只使用训练数据集的一小批次数据，该批次的数据量大小被称为 Batch Size。一次完整的优化迭代包括前向和反向两步：一批次的训练数据先经过神经网络前向得到损失值，根据损失值和链式求导法则反向计算网络中各个参数的梯度，然后基于梯度更新参数。遍历整个训练集，所有数据均参与迭代的过程被称为 Epoch。

3.3 模型与算子

在特定的业务场景上，深度学习需要构建某种特定的神经网络。神经网络通常由多个算子组成，常见的算子包括全连接、卷积、池化、归一化、激活函数等。例如，用于图像分类的卷积神经网络通常由卷积、池化、全连接、归一化和激活函数等算子组成。

4 编译软件栈与深度学习框架

深度学习框架是人工智能领域重要的高性能计算软件，它对底层软硬件栈进行了封装，为深度学习算法研发人员提供简单易用的编程接口。当前，几乎所有神经网络模型的训练与推理任务均基于深度学习框架来完成。寒武纪支持开源深度学习编程框架包括 TensorFlow、Caffe、MXNet、PyTorch、Android NN，其中 Android NN 仅支持寒武纪处理器 IP 核。这些框架的 API 与官方开源框架一致，内部实现则加入了针对寒武纪智能处理器产品的定制。

深度学习框架需要定义张量和算子，张量用于存储计算过程中的数据，算子用于定义各类数学计算。在 GPU 平台上，开发者想在深度学习框架中实现一个算子，一般可以调用 cuBLAS、cuDNN 等英伟达封装好的库函数；或者编写 CUDA 内核函数。

而寒武纪的软件栈与 GPU 有所区别。寒武纪机器学习编程库 CNML 提供了机器学习，主要是深度学习应用的开发所需基础算子，开发者可以便捷地调用这些算子灵活地构造各类深度神经网络模型以及其他机器学习领域的算法，而无须关心寒武纪智能处理器产品的内部硬件资源如何调度。

同时，寒武纪运行时库 CNRT 提供了一套针对寒武纪智能处理器产品的上层编程接口，用于与寒武纪智能处理器产品的硬件之间进行交互和调度硬件资源。用户既可以调用 CNRT 协同 CNML 一起开发深度学习应用，也可以直接基于 CNRT 来高效地运行 Cambricon Neuware——寒武纪端云一体人工智能开发平台所构建的神经网络离线模型。

除此之外，寒武纪 Bang 语言是专门针对寒武纪智能处理器产品设计的编程语言，它支持最常用的 C99 和 C++11 语言的语法特性，并提供了用于编写高性能程序的内置函数接口。此外，Bang 语言还增加了与寒武纪智能处理器产品硬件相关的类型系统，使用这些内置函数和新增类型系统能够充分发挥寒武纪智能处理器的算力和性能。

5 EAST 算法

文字检测是当今计算机视觉的热点问题，最常见的文本检测模型是 ECCV 2016 提出的一种文本检测算法 CTPN，它对水平文字的检测效果很好，但是对于倾斜文字检测效果并不好。而在 CVPR2017 上提出的文字检测算法 EAST，则可以检测任意四边形形状的文本，弥补了 CTPN 的不足。

EAST 算法全名是 Efficient and Accuracy Scene Text，它提出了一个快速而准确的轻量级场景文本检测 Pipeline，解决大多数文本检测算法准确性和效率缺陷。该 Pipeline 只有两个阶段：第一阶段是基于 FCN 模型的文本框检测；第二阶段是对生成的文本框(旋转或矩形)经过非极大值抑制 NMS(Non-Maximum Suppression)得到最终结果，从而免去其他中间步骤，实现端到端训练。

5.1 网络结构

EAST 网络是一个全卷积网络，主要有三大部分：特征提取层，特征融合层，输出层。由于在一张图片中各个文字大小不一，所以需要融合不同层次的特征图，小文字的预测需要用到底层的语义信息，大文字的预测要用到高层的语义信息。

EAST 网络结构图如图 5.1 所示：

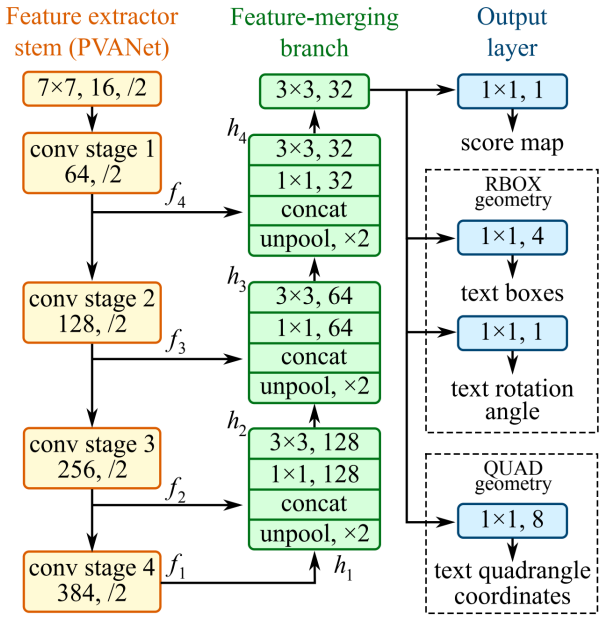


图 5.1 EAST 的网络结构图[7]

第一部分：Feature Extractor Stem (PVANet)

该部分是主干特征检测网络，引入多尺度检测的思想，提取不同尺寸卷积核下的特征并用于后期的特征组合，以适应文本行尺度的变化。作者采用了 PVANet 模型作为主干网络，实际使用的时候也可以采用 VGG16 或者 Resne。

第二部分：Feature-merging Branch

该部分运用了 Unet 的思想，主要是合并第一部分提取的特征，通过池化和 Concat 来恢复尺寸。

其中，特征合并主要通过逐层合并的方式，首先将第一部分对应的特征图输入到一个池化层恢复尺度，然后与当前层特征图进行 Concat，再通过 1x1 卷积来减少通道数，最后利用 3x3 卷积将局部信息融合以最终产生该合并阶段的输出。

第三部分：Output Layer

该部分输出分为 3 类：score_map，RBOX geometry，QUAD geometry

- Socre_map 代表此处是否有文字的可能性；
- RBOX geometry 中的五个通道，分别代表每个像素点到文本框边线的距离，加上一个文本框的旋转角；
- QUAD geometry 中的八个通道，分别代表着每个像素点到文本框任意四边形的 xy 距离。

5.2 Split算子、Sub算子和Concat算子

在 EAST 网络中需要调用 Split 算子，Sub 算子和 Concat 算子完成两个张量之间的相减操作。

Split 算子可以将某个张量在某个维度上进行切分。从而将某个规模较大的张量切分为多个规模较小的张量。Sub 算子可以完成两个张量之间的对位减操作。Concat 算子可以将多个较小规模张量合并成一个较大规模张量。当这三个算子组合运行时实现了将某个较大张量切分后，各自与较小张量相减，最后再将相减结果合并的操作。

在此过程中可以发现 Split 和 Concat 操作均为单纯的内存搬运操作，实际的计算内容只有 Sub 的部分。这提示我们是否可以通过一些优化手段减少这种数据搬运。而这种优化技巧也是本文的核心内容。

6 实验环境与内容

6.1 实验环境

本实验所涉及的硬件平台和软件环境如下：

- **硬件平台**：硬件平台基于前述的 DLP 云平台环境。
- **软件环境**：所涉及的 DLP 软件开发模块包括编程框架 TensorFlow、高性能库 CNML、运行时库 CNRT、编程语言及编译器、运行时库 CNRT。

6.1.1 硬件平台

本实验基于寒武纪 MLU270 智能加速卡，其中搭载的思元 270 DLP 芯片内部集成了 4 个深度学习处理器簇(Cluster)，其中每个 Cluster 包括 4 个智能处理器核及 1 个存储核。每个智能处理器核包括并行的向量和矩阵运算单元、神经元存储单元(NeuronRAM, NRAM)和权值存储单元(WeightRAM, WRAM)，而存储核中则包括共享的片上存储(SharedRAM, SRAM)。其具体结构如图 6.1 所示。该 DLP 硬件以 PCIe 加速卡的形式提供给用户使用，其峰值算力为 128T，支持包括 INT16、INT8、INT4、FP32 及 FP16 等多种不同的数据类型，满足多样化的智能处理需要，兼具通用性和高性能[8]。

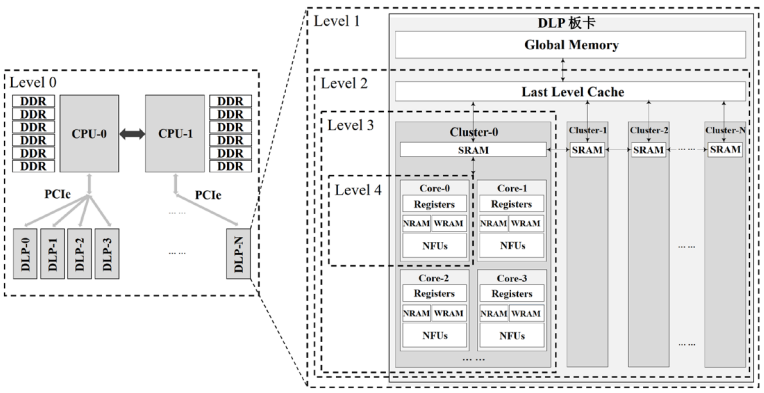


图 6.1 实验所采用的 DLP 硬件架构

6.1.2 软件环境

DLP 硬件的整体软件环境如图 6.2 所示，大致包括 6 个部分：编程框架、高性能库 CNML、智能编程语言 BCL 及编译器、运行时库 CNRT 及驱动、开发工具包及领域专用开发包等。其中，编程框架包括 TensorFlow、PyTorch 和 Caffe 等。DLP 上的高性能库 CNML 提供了一套高效、通用、可扩展的编程接口，用于在 DLP 上加速各种智能算法。用户可以直接调用 CNML 中大量已优化好的算子接口来实现其应用，也可以根据需求扩展算子。智能编程语言 BCL 可以用于实现编程框架和高性能库 CNML 中的算子。DLP 的运行时库 CNRT 提供了面向设备的用户接口，用于完成设备管理、内存管理、任务管理等功能。运行时库作为 DLP 软件环境的底层支撑，其他应用层软件的运行都需要调用 CNRT 接口。除了上述基本软件模块外，还提供了多种工具方便用户进行状态监测及性能调优，如应用级性能剖析工具、系统级性能监控工具和调试器等。

上层智能应用可以通过两种方式来运行：在线方式和离线方式。其中，在线方式直接用各种编程框架(如 TensorFlow、PyTorch、MXNet 和 Caffe 等)间接调用高性能库 CNML 及运行时库 CNRT 来运行。离线方式通过直接调用运行时库 CNRT，运行前述过程生成的特定格式网络模型，减少软件环境的中间开销，提升运行效率。

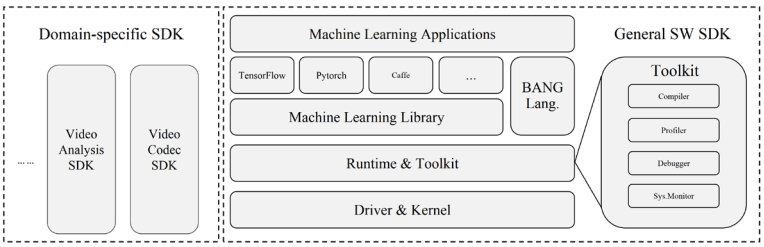


图 6.2 实验所采用的 DLP 硬件的软件环境

6.2 实验内容

本实验主要是在 DLP 硬件上实现并优化以 EAST 网络模型为核心的目标检测应用。针对用户从网络上下载的标准 EAST 模型，经过模型量化并配置 DLP 相关参数，则可以采用 DLP 上的定制 TensorFlow 版本来运行。为了充分发挥 DLP 的计算能力，进一步采用智能编程语言 BCL 实现 EAST 网络模型中的 Split+Sub+Concat 合并算子，并将其添加至 Tensor-Flow 框架中，代替原有的 Split、Sub、Concat 单算子[9]。

具体实验内容包括：

(1) EASTCPU 运行：利用标准的与训练 CKPT 模型，通过定制的 TensorFlow 版本在 CPU 上运行 EAST，得到 Float16 精度的 PB 模型；

(2) EASTDLP 运行：将步骤 1 得到的 Float16 精度的 PB 模型进行 INT8 量化后，通过定制的 TensorFlow 版本在 DLP 硬件上运行；

(3) Split+Sub+Concat 合并算子的 BCL 实现：采用 BCL 实现的 Split+Sub+Concat 合并运算；

(4) 框架算子集成：将用 BCL 实现的 Split+Sub+Concat 合并算子集成到 TensorFlow 框架中。

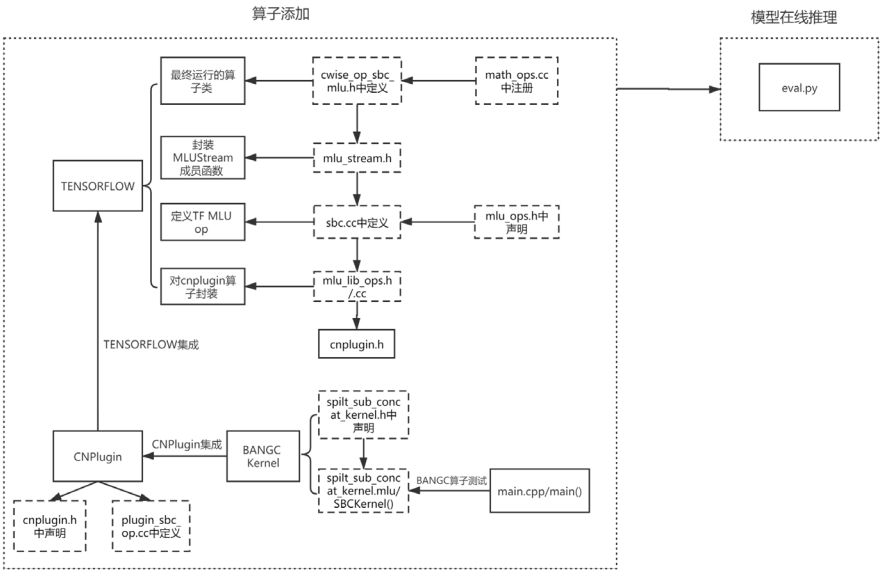


图 6.3 实验内容

7 实验步骤与结果

7.1 实验步骤

如前所述，详细的实验步骤主要包括：EASTCPU 运行、EASTDLP 运行、Split+Sub+Concat 合并算子的 BCL 实现、框架算子集成等。

7.1.1 EAST CPU 运行

为了使得标准的 EAST 网络模型在 DLP 上运行，需要通过运行 CPU 实现，将 CKPT 预训练模型转换成 Float16 精度的 PB 模型。具体步骤如下：

1. **获取开源数据集和 CKPT 模型：**https://github.com/argman/EAST
2. **按照以下目录结构配置数据集和模型：**

数据集目录结构：

```
/home/Cambricon-MLU270/datasets/tensorflow_models/east
            |-- 2015Challenge4_Test_Task1_GT/
            |-- 2015ch4_test_images/
        |-- 2015ch4_training_images/
    |-- 2015ch4_training_localization_transcription_gt/
```

图 7.1 EAST 数据集目录结构

模型目录结构：

```
/home/Cambricon-MLU270/models/tensorflow_models/east/east_icdar2015_resnet_v1_50_rbox
            |-- checkpoint
            |-- model.ckpt-49491.data-00000-of-00001
            |-- model.ckpt-49491.index
            |-- model.ckpt-49491.meta
```

图 7.2 EAST 模型目录结构

3. **DLP 软件环境中已经适配了 EASTCPU 运行环境和 DLP 运行环境，需要进入 tensorflow_models/cambricon_examples/EAST/目录下**
4. **安装推理所需软件包：**一些必备的 Python 安装包在所提供的云平台软件环境中已经安装完成，但对于每个综合实验的特有依赖还需进行单独安装，使用 *pip* 命令完成依赖的安装与升级：
   pip install -r requirements.txt
5. **模型推理和转换：**执行如下命令完成在 CPU 上的推理，并将 CKPT 模型转换为 Float16 精度的 PB 模型：./run_cpu.sh
6. **整理模型路径：**生成的 PB 模型在./cpu_pb 中，需要将得到的 east.pb 拷贝至/home/Cambricon MLU270/models/tensorflow_models/EAST/east.pb

7.1.2 EAST DLP 运行

接下来将上一步得到的 Float16 精度 PB 模型量化为 INT8 精度的 PB 模型。DLP 软件环境提供 fppb_to_intpb 工具来进行模型量化。

1. **进入 tensorflow/cambricon_examples/tools/fppb_to_intpb 目录**
2. **生成数据文件**

按照模型所需要的数据集以及数据集的具体路径来修改 generate_image_list.py 文件中的 image_libs_map，具体如图 7.3 所示，来生成相应的数据文件。

```
// generate_image_list.py

import os
from os.path import isfile, join

max_image_count = 100
image_libs_map = {
      "east":join(os.environ.get( "TENSORFLOW_MODELS_DATA_HOME" ), "east/2015
     ch4_training_images")
      }
if __name__ == "__main__":
  for image_lib, image_path in image_libs_map.items():
    n = 0
    with open("image_list_{}".format(image_lib), "w") as image_list:
      for root, dirs, files in os.walk(image_path):
        for f in files:
          if n >= max_image_count:
            break
          if f.endswith("jpg"):
            image_list.write(join(image_path, f) + "\n")
            n = n + 1
```

图 7.3 EAST 量化生成数据文件

3. **生成量化配置文件**

在进行量化前需设置相应的配置文件以指定量化相关的参数。DLP 软件环境提供了 generate_ini.py 来生成指定模型的量化配置文件。针对 EAST，需要将 generate_ini.py 中的 model_name 进行修改，即指定要生成的配置文件是针对 EAST 的：models_name=['EAST']

执行 DLP 定制化的 PB 模型量化脚本 generate_ini.py 后会生成 config 文件夹，其中包含了 EAST_naive_int8.ini 参数配置文件，该配置文件包含如图 7.4 所示的信息。同时为了规范模型路径，需要修改 save_model_path；

```
// EAST_naive_int8.ini

[preprocess]
mean = 0, 0, 0    #均值,顺序依次为 mean_r、 mean_g、 mean_b
std = 1.0         #方差
color_mode = rgb          #网络的输入图片是 rgb 还是 bgr
crop = 544, 544           #前处理最终将图片处理为 416 □ 416 大小
calibration = east_preprocess_cali       #校准数据读取及前处理的方式,可以根据需求进行自 定义, [preprocess] 和 [data] 中定义的参数均为 calibration 的输入参数

[config]
activation_quantization_alg = naive      #输入量化模式,可选 naive 和 threshold_search , naive 为基础模式, threshold_search 为阈值搜索模式

device_mode = clean                    #可选 clean、mlu 和 origin ,建议使用 clean,使用 clean 生成的模型在运行时会自动选择可运行的设备

use_convfirst = False                    #是否使用 convfirst
quantization_type = int8                 #量化位宽,目前可选 int8 和 int16
debug = False                           #是否为debug模式
weight_quantization_alg = naive          #权值量化模式,可选 naive 和 threshold_search , naive 为 基础模式, threshold_search 为阈值搜索模式

int_op_list = Conv, FC, LRN              #要量化的 layer 的类型,目前只能量化 Conv、FC 和 LRN
channel_quantization = False             #是否使用分通道量化,目前不支持为 True

[model]
output_tensor_names = feature_fusion/Conv_7/Sigmoid:0, feature_fusion/concat_3:0   #输出 Tensor 的名字,可以是多个,以逗号隔开
original_models_path = /home/Cambricon-MLU270/models/tensorflow_models/east/east.pb   #输入pb
save_model_path = /home/Cambricon-MLU270/models/tensorflow_models/east/east_int8.pb     #输出pb
input_tensor_names = input_images:0                        #输入 Tensor 的名字,可以是多个,以逗号隔开

[data]
num_runs = 2      #运行次数
```

图 7.4 EAST 量化配置文件

而提高数据集整体的量化质量。但是对于不存在异常值，数据分布紧凑的情况下，不建议使用该算法，比如网络权值的量化。其中 device_mode 可以设置输出 pb 所有节点的 device，有三种配置方式：clean、mlu 和 origin。其中，mlu 将输出 pb 的所有节点的 device 都设置为 MLU，即都在 MLU 上运行 clean 将输出 pb 的所有节点的 device 清除，运行时根据算子注册情况自动选择可运行的设备；origin 则使用和输入 pb 一样的设备指定。

4. 执行命令

python fppb_to_intpb.py 完成模型量化最终得到 east_int8.pb。

完成 CPU 侧的推理后，接下来需要将其移植在 DLP 上，以加快其推理速度。DLP 软件栈中已经完成了 EAST 的移植，只需要将 MODEL_PATH 指定到 east_int8.pb，执行得到精度和性能数据：./run.sh 16 MLU270 int8 100 1

7.1.3 BANGC 代码实现

1. 设计原则

SBC 合并算子实现在 spilt_sub_concat_kernel.h 和 spilt_sub_concat_kernel.mlu，分别对算子进行定义和实现。在使用 BCL 设计 Split+Sub+Concat 合并算子的时候，需要考虑以下 BCL 设计原则：

(1) 能够处理的数据来源：GDRAM，NRAM，SRAM 中的一种

(2) 能够将数据存放到：GDRAM，NRAM，SRAM 中的一种

(3) 每个 Core 都有一块片上存储空间 NRAM，其大小约为 512KB

(4) 每个 Cluster 上分配一块 SRAM，其大小 2MB，并且单核任务不支持 SRAM 的使用，UNION1 任务和单核任务不支持 SRAM 通信 UNION2 以上任务才支持 SRAM 通信

保证算子正确性的情况下，需要考虑以下四种优化方向：

(1) 对访存进行优化以充分利用片上存储，比如 NRAM 复用

(2) 对计算逻辑进行优化以充分削减计算量，充分利用张量运算器

(3) 充分利用多核并行(计算任务拆分)来提升并行度以及隐藏访存延迟

(4) 充分利用编译器提供的各种编译优化选项，比如 O2 或者 O3 编译优化

2. 设计原理

首先，我们需要分析输入数据的规模，通过开源模型可视化软件 Netron 查看 PB 模型，可以确定整个网络模型的结构，并在 DLPMLU 执行过程中，打印出 input_images 的输出维度为(1, 672, 1280, 3)，即是 Split 的输入维度。

由于 BCL 使用的数据类型是 Half 类型，占据 2 个字节，所以计算出输入规模是 1*x*672*x*1280*x*3*x*2/1024=5040，在充分利用多核并行的情况下，将数据分块到每个 core，每块 NRAM 为 5040/16=315KB，在 512KB 的限制内使用 BangC 实现 Split 的时候，需要注意，输入数据的摆数是 NHWC，C 方向的数据是连续的.如果在 C 方向进行 Split，则需要先转置为 NCHW，或者使用__bang__maskmove 来进行 Split，再去做 Sub，这样会增加计算耗时或者 NRAM 的使用.而最简洁的方法是使用__bang_cycle_sub，对 C 方向进行 cyclesub，省略掉了 Split 和 Concat 部分，大大提升效率，减少 NRAM 的使用。整体设计流程图如图 7.5 所示：

图 7.5 SBC 运行流程

下面以图 7.6 中的代码为例子进行说明，介绍 Split+Sub+Concat 合并算子的接口介绍：

input_data_：输入数据存放的数据地址；

output_data_：输出数存放的数据地址；

batch_num_：Batchsize 的规模。

3. 实现解析

代码实现分为两部分：Kernel 函数实现的 spilt_sub_concat_kernel.mlu 和 cnrt 实现的 main.cpp

```
1    // spilt_sub_concat_kernel.h
2    __mlu_entry__ void SBCKernel(half* input_data_,
3    half* output_data_,
4    int batch_num_)
```

图 7.6 Split+Sub+Concat 合并算子接口

7.1.4 Tensorflow 算子添加

本部分内容将实现的 BANGCkernel 函数集成至 Tensorflow，为 EAST 添加 Split+Sub+Concat 合并算子。添加 MLU 算子主要分为以下几个步骤，步骤之间有承接关系：

1. CNPlugin 编译

如前所述，CNML 通过 PluginOp 相关接口提供了用户自定义算子和高性能库已有算子协同工作(如算子融合)的机制。因此，在完成 Split+Sub+Concat 算子的开发后，可以利用 PluginOp 相关接口封装出方便用户使用的接口(主要包括 PluginOp 的创建、计算和销毁等接口)，使得用户自定义算子和高性能库已有算子有一致的编程接口和模式。

三类函数的具体实现。代码位于 plugin_sbc_op.cc。

2. 算子注册

算子注册在 tensorflow/core/ops 目录下找到对应的 Op 注册文件为 math_ops.cc，:对于 Split+Sub+Concat 合并算子，在 math_ops.cc 中注册。Split+Sub+Concat 合并算子包含一个输入节点 input 和一个输出节点 output，输入输出的数据类型均用 T 表示即该算子的输出与输入数据类型一致。其中，Attr("T:type")表示 T 允许的数据类型为 type，也就是 Ten-sorFlow 支持的所有数据类型。

3. MLUlib

MLUlib 层主要是对 CNML 和 CNRT 接口的封装，该部分代码在 tensorflow/stream_executor/mlu/mlu_api/lib_ops/目录下，需要注意的是 MLULIB 层禁止引用 TensorFlow 的头文件，目前只使用了 tensorflow::Status 类方便做返回值处理。添加 Split+Sub+Concat 合并算子需要对该目录下的 mlu_lib_ops.cc 和 mlu_lib_ops.h 文件进行修改，分别对应实现和定义。

4. MlUops

MLUOps 层主要是使用 MLULib 层封装好的 API 完成算子的实现，代码文件在 tensorflow/stream_executor/mlu/mlu_api/ops 下，需要在目录下的 mlu_ops.h 文件中添加新算子类的声明。同时，需要新建一个 sbc.cc 文件来添加 Split+Sub+Concat 合并算子的实现，其中包含 Creat 方法和 Compute 方法。

Create 方法的职责是将创建好的 baseop 指针调用 base_ops_.push_back(op_ptr);储存起来；Compute 方法的职责在于调用 lib 层的 compute 函数进行计算，目前算子实现为同步方式，因此需要调用 SyncQueue。

5. MLUStream

MLUStream 层负责 MLULOps 算子类的实例化，其接口都定义在 tensorflow/stream_executor/mlu/mlu_stream.h 中。可以把 MLUStream 层的算子分为 2 类：通用模版算子与特例化算子。对于 Split+Sub+Concat 合并算子来说，其属于通用模版算子。

通用模版算子满足以下三个条件：MLU 算子的所有输入均来自 OpKernelContext;MLU 算子的所有输出顺序须与 OpkernelContext 的输出顺序一致；MLUTensor 可以被 CreateM-LUTensorFromTensor 创建，即 MLUTensor 的形状、数据类型与 TensorFlowtensor 一致。

6. MLUOpKernel

MLUOpKernel 层负责对算子抽象定义，其继承了 OpKernel 类，其使用方法与 OpKernel 基本一致.对于每个 MLU 算子均需要实现其构造函数与 ComputeOnMLU 方法。MLUOp-Kernel 实现的主要功能有参数检查、参数处理、输出形状推断及输出内存分配、调用 MLUS-tream 层接口完成算子计算。需要在 cwise_op_sbc_mlu.cc 文件中进行注册和 cwise_op_sbc_mlu.h 实现。

7. BUILD

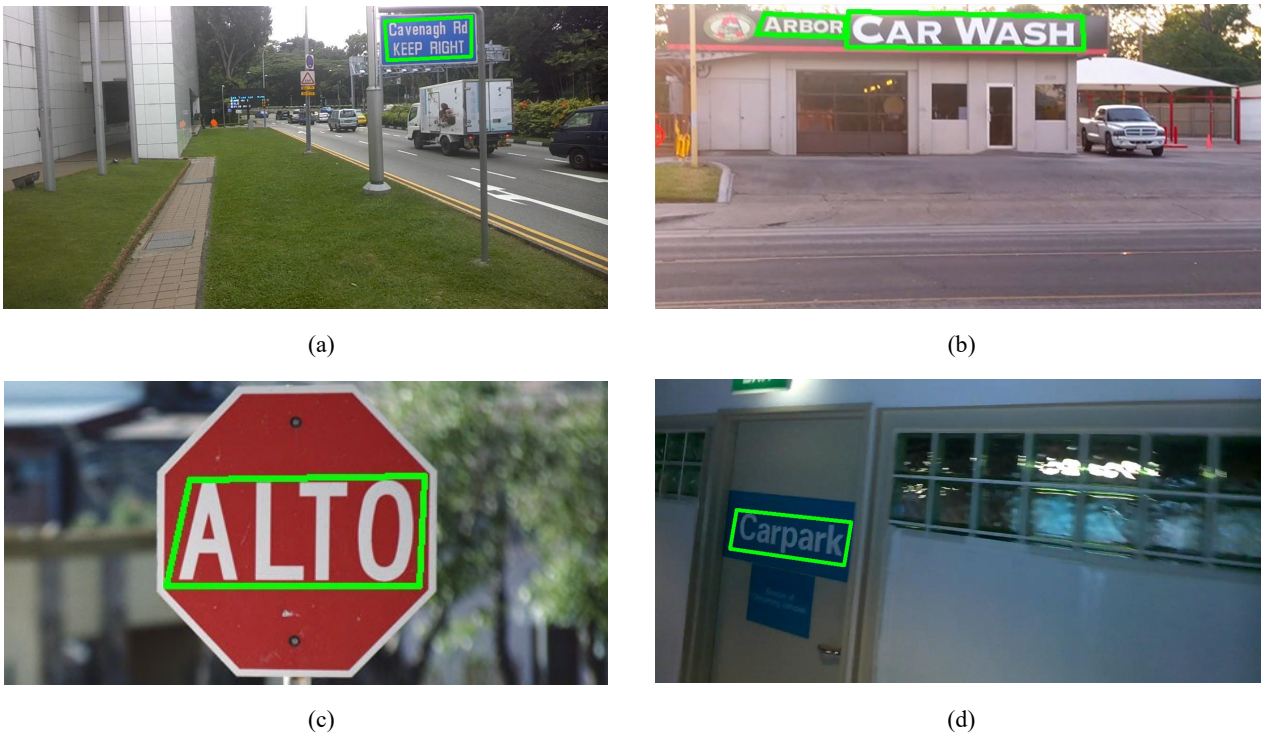
为了将该算子最终集成在 TensorFlow 中，编译时还需要在 tensorflow/core/k-ernels/BUILD 中添加以下信息。

```
// spilt_sub_concat_kernel.mlu
#define HWC_SPLIT ((((HEIGHT*WIDTH/16) - 1) / ALIGN_SIZE + 1) * ALIGN_SIZE)*CHANNELS
#define CHANNELS 3
#define HEIGHT 672
#define WIDTH 1280
#define ALIGN_SIZE 64
#define DATA_COUNT ((CHANNELS) * (WIDTH) * (HEIGHT))

#include "mlu.h"

__mlu_entry__ void SBCKernel(half* input_data_,half* output_data_, int batch_num_){

    int batch_num = batch_num_;

    //struct timeval start;
    //struct timeval end;
    //gettimeofday(&start, NULL);

    __nram__ half split_sub_concat[HWC_SPLIT];
    __nram__ half tmp0[192];

    //多核拆分
    ......

    //循环创建 cycle_sub mask
    for (int i =0;i<192;i++){
        ......
    }

    for (int i = 0; i<batch_num; i++){
        for (int j = 0; j < core_loop; j++){
            //数据拆分至每个 Core 的 NRAM
            ......
            //cycle_sub 代替 split+sub
            ......
            //按顺序拷贝回GDRAM
            ......
        }
        __sync_all();
    }

    //计算耗时
    //gettimeofday(&end, NULL);
    //uint32_t time_usec = (uint32_t)end.tv_usec - (uint32_t)start.tv_usec;
    //printf("Hardware Total Time: %u us\n", time_usec);
    //printf("batch_size: %d us\n", batch_num);
    //printf("core_num: %d us\n", taskDim);

}
```

图 7.7 EAST BANGC 算子实现

7.2 实验结果

实验测试使用了 EAST 模型论文提供的数据集和自行搜集的数据集作为测试图片来源，实验结果如下：



(a) (b)

(c) (d)

图 7.8 实验结果

通过对比数据可知，修改后的模型精度于修改前完全一致，且延时有显著降低。具体数据如下表所示：

表 7.1 模型对比

指标	原始模型 单核	SBC 单核	SBC 16 核
Net FPS	6.067	7.661	12.858
Net 延时	164.8 ms	130.5 ms	77.8 ms
End2End FPS	4.812	5.468	8.444
End2End 延时	207.8 ms	182.9 ms	118.4 ms
Recall 召回率	0.767453	0.767453	0.767935
Precession 精度	0.836306	0.836306	0.836392
Hmean	0.800402	0.800402	0.800703

8 总结

寒武纪思元 270 是一款新型 DLP 深度学习处理器．与 GPU 相比，其专门面向深度学习计算，设计了专用计算单元，且基于寒武纪的软件栈与 GPU 有所差异。为深入挖掘寒武纪的性能和优化方法，本文依据现有的文本检测的深度学习模型 EAST，基于搭载有寒武纪思元 270 DLP 芯片的深度学习加速卡 MLU270，加速模型计算速度，并将用 BCL 实现的 Split+Sub+Concat 合并算子集成到 TensorFlow 框架中的优化手段，通过减少数据搬运，实现了更高效的识别和更低的延时。

基于本文实验中所展现出的出色成绩，也期待寒武纪之后能推出更新更强的 DLP 深度学习处理器，以及其能更广泛地在视觉、语音、自然语言处理以及传统机器学习等场景中的应用，成为“中国芯”逆袭力量的代表。

致谢

写到这里，我不禁回顾了自己三年的大学学习生涯。个人认为，《研究方法与前言(计算机)》是计算机专业所有课程中极为重要的一门。就像童老师第一节课中强调的“Search、Search、再 Search”，在本课程中，通过提出问题、研究探索、得出结果的过程，培养了我们研究问题的思维方式。与此同时，可以掌握不少研究与学习的方法。以我自身为例，通过老师讲解、同学间互相探讨，我逐步提高了发现问题、分析问题和解决问题的能力，为我以后的学习与研究奠定了基础；通过上台报告，提高了我查阅文献及表达能力；同时，通过撰写这篇报告，我也提高了文字编辑、排版等方面的能力。

普通人的记忆力是有限制的。大学本科的一切课程，在多年后回想其中的知识点，可能是模糊不清的。但是在学习中掌握的学习技巧、方法，会流淌在血液中，帮助我们提高各方面的素养，可以更快更好的学习其他知识。我想这正是学习的意义所在。

最后，我想感谢童老师的认真教授，使我掌握了课程的知识点，并在《实用计算机英语》的课后和我交流 ChatGPT 的前景未来，耐心地解答了我的困惑。也感谢同学们在本学期中，对我各方面的支持。在大家的帮助下，我顺利地完成了《研究方法与前言(计算机)》此课程的学习。

参考文献

[1] 白志程,李擎,陈鹏,郭立晴.自然场景文本检测技术研究综述[J].工程科学学报,2020,42(11):1433-1448.DOI: 10.13374/j.issn2095-9389.2020.03.24.002.

[2] 余超. 基于深度神经网络的自然场景文本检测研究[D].南昌大学,2021.DOI:10.27232/d.cnki.gnchu.2021.001449.

[3] 徐本朋. 基于深度学习的场景文本检测方法研究[D].安徽大学,2020.DOI:10.26917/d.cnki.ganhu.2020.000438.

[4] 冷莉,邹威.面向自然场景的多语言文本特征自动检测研究[J].自动化与仪器仪表,2021(12):24-27.DOI:10.14016/j.cnki.1001-9227.2021.12.024.雷思磊.

[5] 王哲,冯晓辉.AI芯片:加速智能时代的发动机[J].人工智能,2018(02):20-33.DOI:10.16453/j.cnki.issn2096-5036.2018.02.002.

[6] 鲁蔚征,张峰,贺寅烜,陈跃国,翟季冬,杜小勇.华为昇腾神经网络加速器性能评测与优化[J].计算机学报,2022,45(08):1618-1637.

[7] Zhou X , Yao C , Wen H , et al. EAST: An Efficient and Accurate Scene Text Detector[J]. IEEE, 2017.

[8] 杜子东.寒武纪:智能处理器和基准测试集[J].人工智能,2018(02):72-81.DOI:10.16453/j.cnki.issn2096-5036.2018.02.006.

[9] 李玲,郭崎,陈云霁. 智能计算系统实验教程[M]. 北京: 机械工业出版社, 2021.