目录

边缘智能联合研究

1 摘要

智能世界带来了丰富的应用，产生了海量的连接和数据。边缘计算兴起，未来超过 70%的数据和应用将在边缘产生和处理。边缘和移动端设备受场景约束，处理能力和性能的提升受到限制，需要与云协同。设计一个边缘智能框架，需要考虑的因素很多，主要包括云边协同计算的框架选择、软硬件加速和对 AI 模型的优化。

2 边缘框架对比及使用测试

2.1 几类边缘计算目的差异

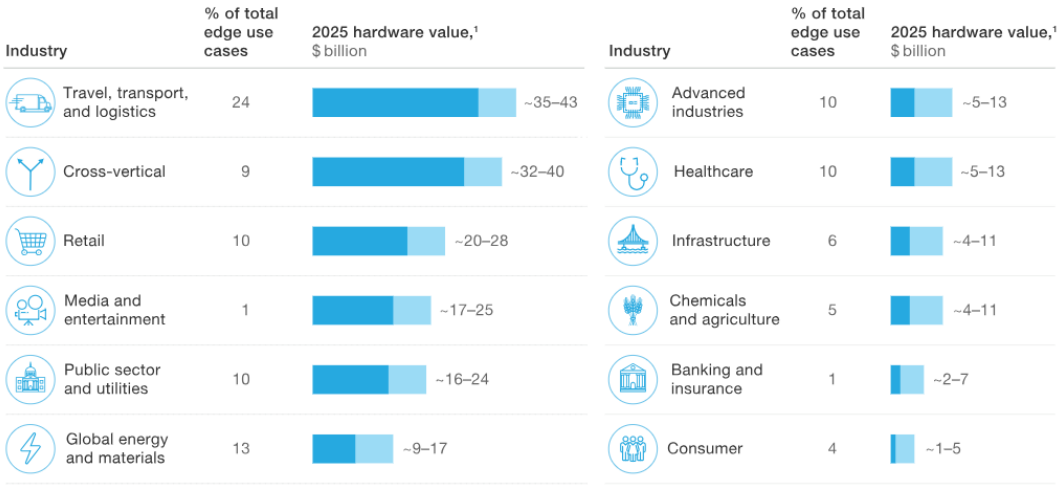
1、 Fog computing (FC)：利用设备和云之间的任意节点的 Fog Computing nodes (FCNs)，例如路由器、交换机等进行分布式计算。
2、 Mobile/Multi-access Edge Computing(MEC)：使无线电网络基站等具备计算和存储能力
3、 Cloudlet(CC)：”data center in a box”，在靠近设备端的虚拟机提供高带宽低延迟的访问。

2.2 几类的需求特征

	FC	MEC	CC
位置	靠近终端设备，密集、分布式	无线电基站	本地/户外设备安装
设备	路由器，交换机，AP，……	基站处的服务器	小尺寸数据中心
访问媒介	WiFi, LTE, MQTT, ……	WiFi，LTE，…..	WiFi……
实时交互能力	高	中	中
功耗	低	高	中
计算能力	中	高	高
识别应用场景	中	高	低
覆盖率	低	高	低
服务器密度	中	低	高
成本	低	高	中
活跃用户	高	中	中

2.3 边缘计算应用占比

Edge computing represents a potential value of $175 billion to $215 billion in hardware by 2025.



Industry	% of total edge use cases	2025 hardware value,[1] $ billion
Travel, transport, and logistics	24	~35–43
Cross-vertical	9	~32–40
Retail	10	~20–28
Media and entertainment	1	~17–25
Public sector and utilities	10	~16–24
Global energy and materials	13	~9–17
Advanced industries	10	~5–13
Healthcare	10	~5–13
Infrastructure	6	~4–11
Chemicals and agriculture	5	~4–11
Banking and insurance	1	~2–7
Consumer	4	~1–5

Total: ~$175 billion–$215 billion

[1]Hardware value includes opportunity across the tech stack (ie, the sensor, on-device firmware, storage, and processor) and for a use case across the value chain (eg, including edge computers at different points of architecture).

POWER

MARKET INSIGHTS FOR 5G / EDGE / IOT

$700 BILLION BY 2030 (ANNUAL TSP 5G REVENUE)

(ERICSSON AND ARTHUR D. LITTLE REPORT)

Industry	MRKT %	CAGR %	Primary Application	Other Applications
Healthcare	21%	75%	Telemedicine	Hospitals
Manufacturing	19%	76%	Automation, A/R	Robotics
Energy / Utilities	12%	67%	Power Generation	Smart Buildings
Automotive	12%	71%	Autonomous Cars	In-Car Entertain
Public Safety	10%	78%	Surveillance	Public Safety
Media / Entertain	10%	86%	Gaming	Advertising
Financial Services	5%	76%	Banking	Securities
Public Transport	5%	65%	Mass Transit	Goods Delivery
Retail	4%	76%	E-Commerce	Stores
Agriculture	2%	85%	Farming, Livestock	Fishing, Hunting

2.4现今边缘计算硬件设备

1. Facebook Yosemite V3

基于 Xeon D（集成 10GbE SoC），提供紧凑、高度模块化的边缘节点。

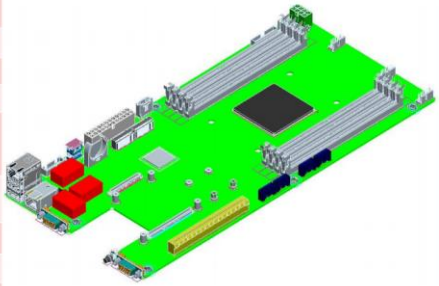
Delta Lake Server Web Server Configuration

Delta Lake Server with 1OU Expansion
- Up to 4x 22110 M.2 Form Factor Devices
- Supports Multi-Host NIC configuration

Delta Lake Server with 2OU Expansion
- Up to 6x 22110 M.2 Form Factor Devices
- Dedicated BMC
- Dedicated Single Host NIC

2. OpenEdge with Ampere

	Hawk
CPU	• 1x Ampere eMAG 8180 CPU (TDP ~110-125W), 32x ARMv8 64-bit CPU cores at 3.30 GHz with Turbo • 1x Ampere eMAG 8140 CPU (TDP ~75W), 16 x ARMv8 64-bit CPU cores at 3.30 GHz with Turbo
Form Factor	• Half Width (1U)
Chassis	• openEDGE, Open19
Memory	• 8 x DDR4 2667 RDIMM (1DPC)
Network	• OCP Mezzanine v2 (Conn. A and Conn. B) supporting 10/40/100 GbE NIC • 1 x 1 GbE (RJ45) Onboard
Storage	• 2 x NVMe M.2 (PCIe x4) • 4 x SATA (Gen3)
PCIe Expansion	• One PCIe slot (x16)
Other I/Os	• VGA, USB 2.0
Power	• Power Shelf, ATX
Mgmt.	• 1 G RJ45, slimline (openEDGE)



3. Nvidia Jetson 系列

在类似树莓派等的小尺寸板上提供 CUDA GPU 来支持机器学习应用。

4. Supermicro IP65 Edge Server

超微发布的 IP65 级防尘防水的服务器（机箱），基于 Xeon D 或 Xeon SP，设计安装使用于基站等位置，提供户外的散热、加热能力等

2.5 常见边缘计算研究方向

1. 联合学习/联邦学习

i. Adaptive Federated Learning in Resource Constrained Edge Computing Systems

ii. Distributed Deep Learning Model for Intelligent Video Surveillance Systems with Edge Computing

2. 边缘计算框架

i. EdgeWise: A Better Stream Processing Engine for the Edge

3. 任务分配/平衡算法

i. Edge server placement in mobile edge computing

ii. Balancing on the edge Transport affinity without network state

4. 可靠性

i. Wireless Edge Computing With Latency and Reliability Guarantees

5. SDN 及网络模型

i. A Multi-Clustering Approach to Scale Distributed Tenant Networks for Mobile Edge Computing

ii. Modeling The Edge: Peer-to-Peer Reincarnated

6. 安全

i. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone

7. 协议

i. Publish-Pay-Subscribe Protocol for Payment-driven Edge Computing

2.6 边缘计算框架对比

1. Full Stack & IaaS

Project	Foundation	Key Participators	Layer	Segment/Focus	MANO	SDN	Latest version	Infra	Code Repo
Akraino	Linux Foundation	AT&T, Intel, ARM, Nokia, Ericsson, Dell, Red Hat, Juniper, WRS, etc.	Umbrella, Full Stack	All-in-one edge stack	N/A	N/A	N/A	Openstack, K8S	http://gerrit.akraino.org
StarlingX	OpenStack Foundation	Wind River, Intel, Huawei, Ericsson, China Unicom, etc.	IaaS	Industrial IoT and MEC	ONAP	ODL	1.0	OpenStack	https://git.starlingx.io/cgit
Airship	OpenStack Foundation	AT&T, SKT, Intel, Mirantis, etc.	Deployment	Openstack on Kubernetes	ONAP/Tacker	Calico	0.1	OpenStack/K8S	https://git.airshipit.org/cgit
CORD	Linux Foundation	AT&T, SK Telecom, Verizon, China Unicom and NTT, etc.	IaaS	MEC for residential, enterprise & mobile	XOS	ONOS	6.0	OpenStack/K8S	https://github.com/opencord
vCO	Linux Foundation	Red Hat, China Mobile, etc.	IaaS	MEC for residential, enterprise & mobile	ONAP/Tacker	ODL	2.0/3.0	OpenStack	No code repo yet. Just POC

28

1. PaaS & IoT

Project	Foundation	Key Participators	Scope	Layer	Segment /Focus	Latest version	Code Repo
EdgeX Foundry	Linux Foundation	Dell, Vmware, etc.	Common framework for Edge solutions (SDK).	PaaS	Industrial IoT	3.0 (4.0 expected in April 2019)	Go: https://github.com/edgexfoundry/edgex-go Java: https://github.com/edgexfoundry
OpenEdge	N/A	Baidu, etc.	Open edge computing framework	PaaS		0.1.2	https://github.com/baidu/openedge
KubeEdge	CNCF, Linux Foundation	Huawei, etc	Extend native containerized application orchestration capabilities at Edge	PaaS		0.2	https://github.com/kubeedge/kubeedge
Azure IoT Edge	N/A	Microsoft	Internet of Things (IoT) service that offload task to edge	PaaS	IoT	1.0.8-dev	https://github.com/Azure/iotedge
ioFog	Eclipse Foundation	Edgeworx, etc.	Edge computing platform through microservice at edge	PaaS	IoT	2.0/3.0	https://github.com/ioFog/iofog.org
Eclipse Kura	Eclipse Foundation	Eurotech, Rad Hat, Comtrade, etc.	Platform for building IoT gateways, enabling remote management & app deployment	PaaS	IoT	4.0	https://github.com/eclipse/kura/

2.7 几个边缘框架功能对比

	Edge X	K3S	KubeEdge	StarlingX	OpenEdge
云边协同	不支持	不支持	支持	支持	支持
基于 K8S	否	是	是	否	否
资源需求	中	小	最小	较大	较大
部署复杂度	简单（docker）	简单（现有 K8S 应用框架） 复杂（无现有 K8S 应用）	简单（现有 K8S 应用框架） 复杂（仍不成熟，鲁棒性较差）	复杂	复杂
去中心化部署	否	否	是	否	否
MQTT 支持	是	是	是	是	是
容器编排管理支持	否	是	是	是	否

KubeEdge 官方 Q&A：

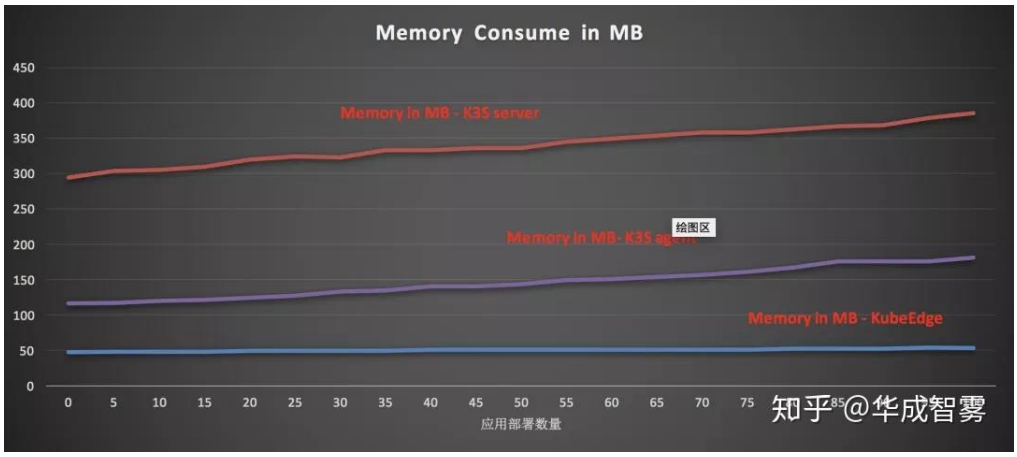
1. 简单对比 KubeEdge 与国内外的主流同类型平台产品

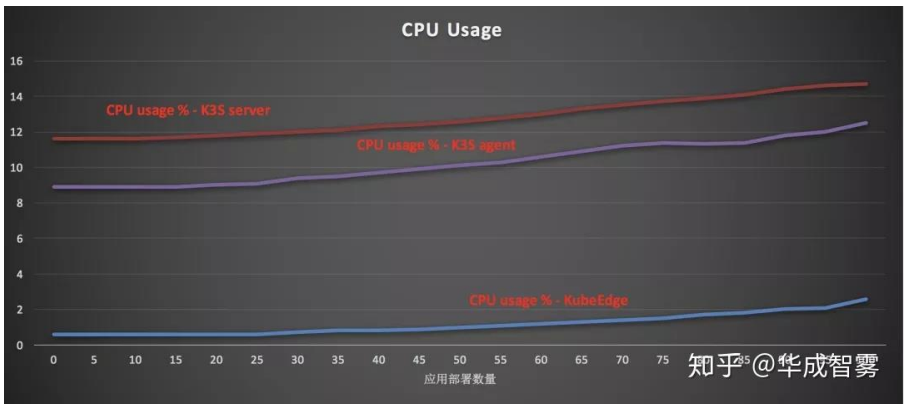
EdgeX Foundry 偏重于端侧设备的管理，不具备云上对边缘端的应用和设备的管控、云边协同等智能边缘系统的能力。K3s 选择的是在边缘运行整个 K8s 集群的方案，不具备云边协同的能力；其次 K3s 虽然对 K8s 做了轻量化，但整体资源要求仍然较高，无法运行在 IOT Hub、工业网关等小型设备中。

2. KubeEdge 的 API 调用方式，是否计划提供 sdk 调用方式？

由于 KubeEdge 不屏蔽 K8s APIserver，涉及原生功能的 API 可以直接使用 K8s client-go 调用，KubeEdge 扩展的 API，计划于 1.3 版本（2020 年 Q1）提供 SDK。

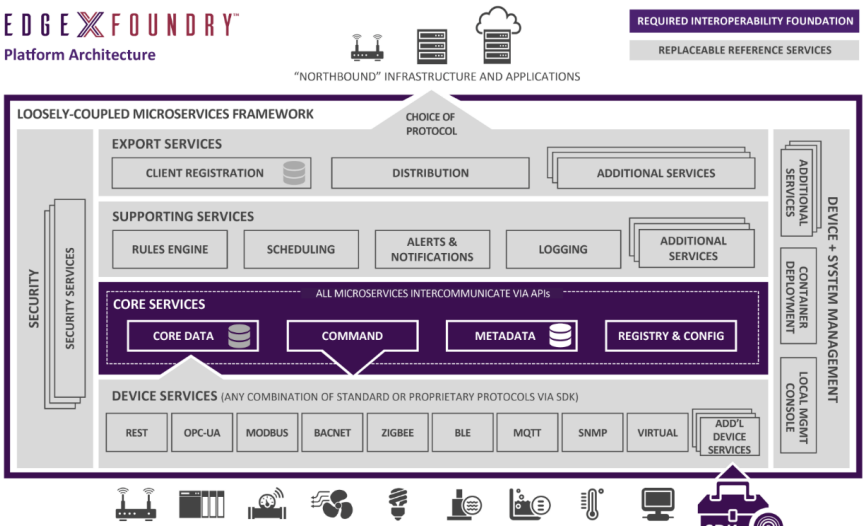
KubeEdge、K3S 资源占用测试对比





KubeEdge EdgeX 使用体验对比

1. EdgeX

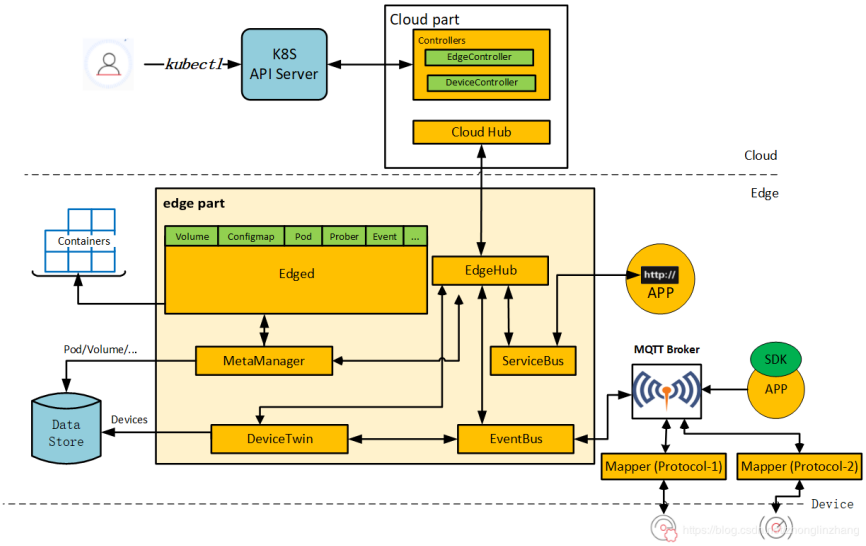


EdgeX 架构图

EdgeX 可基于 docker 部署，注重于端侧（南侧）设备管理，使用 REST 接口与云端（北侧）进行通讯。框架本身并不提供内建的云边通信协作接口，需要使用其他服务或自行开发。测试的部署使用 docker-compose 配置文件，框架服务运行于 docker 中，经过测试在部署了 core、scheduler、redis、mgmt、notification 等核心服务和三个基础的设备应用服务时，内存占用约 1200M（redis 服务 950M，其他组件 200M），CPU 使用率约 3-4%。（前文 K3S 与 KubeEdge 测试未提到测试环境，不具备直接可比性）默认使用 Redis，内存需求较大，可使用 mongo，其他数据库目前尚未官方支持。

提供 go 与 C 的 SDK，IoT 设备配置文件较 KubeEdge 简洁明了，提供较强的历史数据持久存储管理能力，可靠性强，适合工控等要求。

2. KubeEdge



KubeEdge 架构图

基于 K8S 精简而来，相较于 EdgeX 注重 IoT 设备的数据采集与管理，主要面向于云端与边缘端容器的管理，似乎不具备设备的持久历史数据管理支持。设备配置基于 Device Twin 格式的 yaml 文件，与 IoT 设备的通信管理接口较为繁杂。

提供了基于 K8S API Server 的边缘容器和 IoT 设备的各项管理能力，在边缘节点离线的

情况下边缘设备仍能够持续运行，提供了完善的边缘容器的管理能力。目前无 SDK 提供，仅支持 K8S 原生 Go 支持，编程接口较为缺乏。

项目仍属发展较早期阶段，文档更新频繁，导致了目前文档混乱不清的问题。同时非常依赖于 K8S 环境配置，容易出现错误。出现错误时报错不清，没有报错内容，无法快速确定错误源头。同时依赖的 K8S 与 Go 环境均受墙影响，会对部署环境网络要求极高，出现由于网络原因导致的错误时难以非常排查，框架也无法处理由于网络问题部署出错的情况。出现问题甚至可能不报错，需要仔细查阅日志文件。

参考文献

[1] https://146a55aca6f00848c565-a7635525d40ac1c70300198708936b4e.ssl.cf1.rackcdn.com/images/bc4391e72152428cc896b3a55350bd735adb7736.pdf

[2] https://146a55aca6f00848c565-a7635525d40ac1c70300198708936b4e.ssl.cf1.rackcdn.com/images/5867d6636fed6209a55ec81ec54a6fa89776ab2e.pdf

[3] https://www.cnblogs.com/wsjhk/p/12103998.html

[4] https://zhuanlan.zhihu.com/p/65131381

[5] https://arxiv.org/pdf/1911.02794.pdf

3 基于深度学习的软硬件协同设计与方法

3.1 国内外研究现状

当前，深度学习基本已进入成熟期与瓶颈期，成熟是指业界很多公司都已将深度学习融入进自己的业务，而瓶颈是指大部分由上世纪 80-90 年代打下的深度学习理论基础已基本变现[1]，而新的突破性进展仍未出现，近期也可以看到很多在业界的老师回归高校[2]。

因此，在当前时间节点下，对深度学习的软硬件协同设计进行一次较系统的梳理是非常恰当的。如图 1 所示，当前的深度学习软硬件栈实际形成了类似 TCP/IP 栈两头宽-中间窄的形状。算法研究者与工程师更偏向于 ML 应用至 ML 模型的三层内容；而系统方向的研究者以及编译、硬件架构的研究者则更关注于底层的内容。

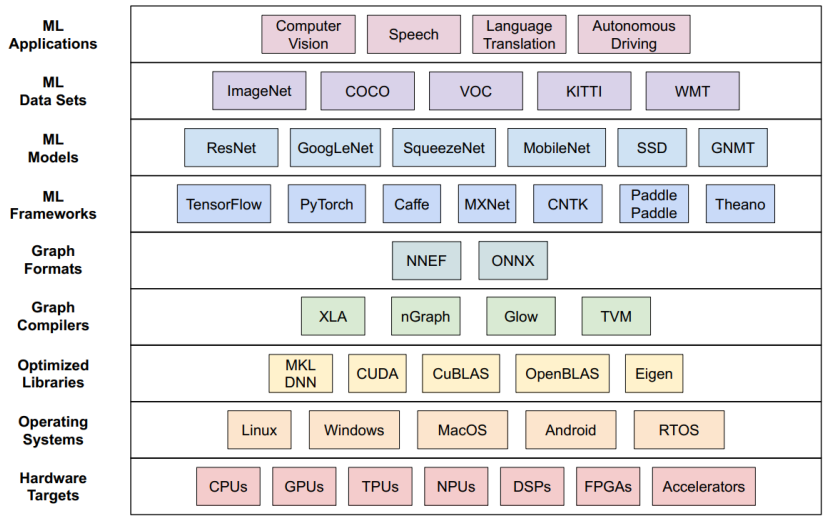


图 1 深度学习推断阶段的软硬件栈[3]

当前的深度学习算法大部分都拥有两个阶段：训练阶段(Training)与推断阶段(Inference)。训练阶段，一般通过大批量数据根据算法规则来更新模型参数，考虑到训练阶段的计算量较大且持续时间久，往往考虑在服务器端、云端进行；推断阶段，由于应用场景的不同，考虑到成本、功耗、大小等因素，运行平台也会有所差异[4]。

在 ML 算法的研究阶段，算法研究者更关注于特定领域中的某些指标（如准确度，IOU 等等），往往对实际应用场合中的部署限制不知情或不关心；而对于云服务提供商或者边缘计算的应用来说，这都是难以接受的。所以，目前也有很多的研究更关注于如何更高效地、更高能效地进行模型的训练与推断，如量化、剪枝、压缩等等[5-6]；而一些比较主流的 ML 框架如 Tensorflow 和 Pytorch 本身也提供了对模型量化的支持。

在 ML 模型的部署阶段，考虑到 ML 模型的快速迭代，想要在特定平台上支持较长时间的各类模型的快速与高效部署实际是非常困难的，即使是专业人士进行手工优化也是得不偿失的。所以，当前主流考虑使用类似 XLA 和 TVM 的深度学习编译器[7]，起到类似 LLVM 的作用，将不同的深度学习模型自动编译成对应不同平台的高效的执行代码，这一块也是目前的热点之一，实际也还面临着非常多的问题。

而对于最底层的硬件架构，经历了接近 5~8 年的研究与实践，最终留下了 5 种特定的硬件及其对应架构：（1）GPU：是目前训练阶段的主流选择，但是利用率问题一直仍是一个热点，并且也在近几年推出了嵌入式 GPU，以应对边缘计算（2）TPU：原始版本[8]的计算部分主要采用脉动阵列架构，现公开的版本（v2/v3）公布了更多模块级的细节，主要应用于 Google 自己的云端[9]（3）类 SIMD 架构：最先由寒武纪的 Diannao 引起关注[10]，是主流架

构之一，也出现在特斯拉芯片中[11]（4）Graphcore: 类似众核架构，公布性能接近顶尖[12]。(5) Habana:已被 Intel 收购，实际总体架构某种程度上类似 TPUv2/v3 以及 DaDiannao[13]。

3.2 基本概念

3.2.1 优化方法

考虑到内容实在太多，所以不在此展开介绍。由于 ML 中的运算实际都可转化为矩阵运算，实际对于 ML 中的运算本身分为两种思路进行计算：（1）转化为矩阵，那么只需考虑对矩阵的优化方法（快速矩阵乘法：FFT、Winograd）以及对于 ML 场合下的矩阵本身具有的特性进行相应优化的权衡（稀疏性、概率分布等等引入了 QR 奇异值分解等）。（2）仍然考虑定义中的计算方式，一般采用 Loop Tiling、Unroll、Exchange 等利用计算的局部性与并行性，结合实际运行平台，考虑各种并行（多核并行，SIMD，多节点并行，多 stream 并行……）以提高设备的利用率。

此外，由于 ML 模型本身具有的某些特性（模型的精度损失，权重的稀疏性和分布特性，实际可通过训练来人为改变），引入压缩、剪枝和量化，以减少运行代价。

以上说的都只是针对于给定某种特定硬件架构的前提下，如果考虑 FPGA，实际可以做进一步优化（不知道是幸运还是不幸），也就是所谓的软硬件些协同设计，设计者可以根据所需要达成的指标（如精度、FPS 等等），综合考虑架构设计、模型设计以及模型的剪枝量化压缩等优化方法，通过添加一定的约束，在有限的时间内按某种算法搜索整个设计空间来设计对应的硬件架构以及网络模型（现在一般考虑用 NAS 来做，有点类似 TVM 的用 ML 来做 ML 的优化的感觉）。

3.2.2 评价基准与测试集

目前主流考虑用 MLPerf 中的模型和评价标准来进行评估，本质上是对于不同的 ML System，用不同的模型进行评估，主要评估性能（更偏向吞吐量和时延）；前者本质上是所谓的 macro benchmark，而对于更细粒度的评估（即各种不同类型的算子），即 micro benchmark，考虑到各软硬件系统的差异性，很难在同一尺度下进行衡量[14]。

在 MLPerf 出现前，主要考虑使用 Roofline[15]模型进行评估。Roofline 模型是一种直观的性能评估模型，用于评估和探索在给定硬件系统（已知峰值计算性能和理论最大带宽）的情况下，对于各种应用所能达到的峰值性能与优化方向，典型 Roofline 模型，如图 2 所示：

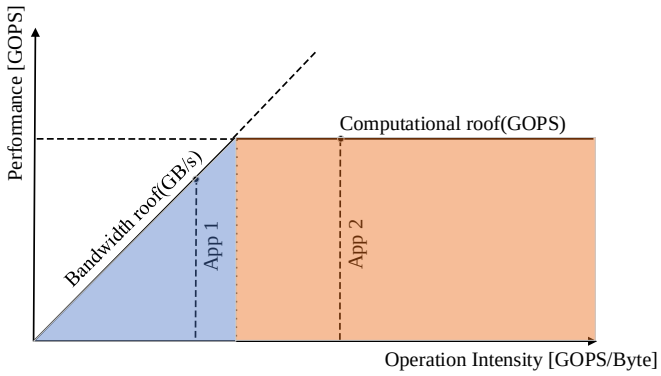


图 2　Roofline 模型

其中，计算上限（Computational roof）表示给定硬件系统所能达到的理论峰值计算性能，带宽上限（Bandwidth roof）表示给定硬件系统与内存交互的最大带宽。横坐标计算密度

（Operation Intensity）表示平均从内存读取或写入单位数据所参与的计算量，一般用总计算量除以总访存数据量得到；纵坐标表示系统以及各应用所能达到的性能。
应用 1 落在带宽受限区域（蓝色区域），如果应用 1 的总计算量与访存量不发生改变，最大只能达到带宽上限所限定的性能。应用 2 落在性能上限的区域（橘色区域），此时应用 2 可以达到系统的最大性能。实际应用中，希望应用能够落在橘色区域，且越靠近右上方的计算上限越好，主要有两个原因：（1）越向上靠近性能上限，说明实际性能越接近最大性能，系统利用率越高。（2）越靠近右方，说明计算密度越大，即单位内存数据交换对应的计算量越大，能效越好。

3.3 深度学习的软硬件优化

3.3.1 软件相关

深度学习框架与运行时优化

主要考虑 Tensorflow 和 Pytorch，两者之前最大的区别在于：Tensorflow 需要在运行前先完成整个计算图的建立（静态图），Pytorch 在运行时才会建立图，反向时销毁（动态图）；实际现在应该相差不大，TF2 目前也已支持动态图，但 Pytorch 给人的用户体验仍然更好。当前的优化主要集中于多节点训练以及端设备上的运行时优化。

编译优化

由于个人对这部分不是非常了解与深入，实际可参考唐杉博士的相关文章[7]，这里简单的对内容进行概括。不管是 TVM 还是 XLA，实际都是希望能够让日益增长的 ML 模型有效且快速地映射到不同的后端设备执行而已，近两年的工作主要考虑在不同硬件平台上的算子融合、负载均衡等问题，也是当前的热点之一。

库优化

cuDNN、CUDA 之类的库大部分是针对特定硬件，由各平台厂商进行维护与更新。深度学习相关大部分的库，主要针对于矩阵运算进行优化，实际也是考虑不同规模下，选择合适的优化手段来进行快速的矩阵运算，一般包括 FFT、Winograd 等等。实际在公司内部，也会有专业人士进行细粒度的优化或建立公司的优化库，一般应用只需要在 API 层上优化[16]。

3.3.2 硬件相关

架构优化

硬件架构目前已经比较成熟，最初是由中科院计算所陈天石等人在 ASPLOS'14 提出的 DianNao 架构[10]，应用 Loop Tiling 和 Unroll，给出了类似 SIMD 的计算架构，以及定点 16 位量化对准确度损失不大的结论。在 FPGA'15，北大的张宸、UCLA 的 Jason Cong 等人在前者的基础上引入了 Roofline 模型来对深度学习加速器进行性能评估[17]。在 FPGA'16 上，清华大学的姚松和汪玉等人总结了在嵌入式 FPGA 上部署类 SIMD 架构的 CNN 加速器的整体流程[18]。在 FPGA'17 上，斯坦福的韩松以针对 LSTM 的软硬件协同设计拿下了 best paper，实际是第一次通过量化剪枝压缩，以及相应的硬件架构设计使得基于 FPGA 的 DL 加速器在性能上超过了对标 GPU 的水平[5]。在 ISCA'17，Google 给出了 TPU v1 的整体架构以及使用 Roofline 模型分析了对于 Google 数据中心的主流应用 TPU 的对应性能[8]（对应脉动阵列架

构)。其他还有很多，这里不再罗列，个人感觉在 DL 模型的部署方面，2014~18 年间在硬件架构上的工作都还是非常有价值的，后面基本在意料之中了。

对于一些其他的优化方法，在 2.1 节中也已基本涵盖，只是硬件会考虑到计算的依赖性、局部性、并行性来设计对应的 Load-Compute-Store 流水和相应模块。

3.4 近阶段趋势

编译优化与运行时优化

运行时优化主要考虑端设备在运行过程中的负载均衡。虽然编译优化对主流架构做到很高的优化程度，但考虑到当前模型的发展趋势（又回到了类似 VGG16 的大规模权重，同时还伴随着更加复杂的网络结构，如 MLPerf v0.7 中的 BERT），也是一个非常有挑战性的问题。

软硬件协同优化与网络架构搜索（NAS）

在 2.1 节中也已经涉及了部分，考虑到网络结构的多变性，目前的趋势是通过 AutoML 类似的自动网络架构搜索方法，通过提前设定元结构来搜索设计空间，以得到满足需求的模型；也有用这种方法来同时搜索硬件架构的设计空间的，本质上是一样的[19]。

超算

最近半年，诸如 Google、Nvidia、AMD 等都在做超算（虽然目前很多是面向 AI 的超算，但很有可能会入侵到传统的科学计算领域），实际也是因为本身 AI 的算子与一般科学计算的算子差别不大，而且考虑到他们本身在云和数据中心场合下的占有率，也就不足为奇[9]。

参考文献

[1] https://github.com/exacity/deeplearningbook-chinese《深度学习》花书

[2] https://www.zhihu.com/question/404733616 如何看旷视南京负责人魏秀参跳槽高校工作?

[3] Reddi V J, Cheng C, Kanter D, et al. MLPerf Inference Benchmark.[J]. arXiv: Learning, 2019.

[4] https://zhuanlan.zhihu.com/p/77223000 AI 芯片相关热词解析

[5] Han S, Kang J, Mao H, et al. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA[C]. field programmable gate arrays, 2017: 75-84.

[6] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding[J]. arXiv preprint arXiv:1510.00149, 2015.

[7] https://zhuanlan.zhihu.com/p/29254171 Deep Learning 的 IR“之争”

[8] Jouppi N P, Young C S, Patil N, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit[C]. international symposium on computer architecture, 2017, 45(2): 1-12.

[9] Jouppi N P, Yoon D H, Kurian G, et al. A domain-specific supercomputer for training deep neural networks[J]. Communications of the ACM, 2020, 63(7): 67-78.

[10] Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[C]. ASPLOS, 2014, 49(4): 269-284.

[11] https://www.hotchips.org/hc31/HC31_2.3_Tesla_Hotchips_ppt_Final_0817.pdf

[12] https://www.graphcore.ai/

[13] Luo T, Liu S, Li L, et al. Dadiannao: A neural network supercomputer[J]. IEEE Transactions on Computers, 2016, 66(1): 73-88.

[14] Tao J H, Du Z D, Guo Q, et al. Bench ip: Benchmarking intelligence processors[J]. Journal of

Computer Science and Technology, 2018, 33(1): 1-23.
[15] Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures[J]. Communications of the ACM, 2009, 52(4): 65-76.
[16] Ryoo S, Rodrigues C I, Baghsorkhi S S, et al. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA[C]//PPoPP. 2008: 73-82.
[17] Zhang C, Li P, Sun G, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks[C]// FPGA. 2015: 161-170.
[18] Qiu J, Wang J, Yao S, et al. Going deeper with embedded fpga platform for convolutional neural network[C]//FPGA. 2016: 26-35.
[19] Hao C, Chen Y, Liu X, et al. NAIS: Neural architecture and implementation search and its applications in autonomous driving[J]. arXiv preprint arXiv:1911.07446, 2019.

4 神经网络模型的压缩与优化

4.1 研究背景

当前，“边缘计算”越来越重要，但神经网络模型的性能提高往往意味着巨量的参数和计算量。专用硬件平台可以很好的解决计算与存储的双重需求，但目前还不成熟，存在些亟待解决的问题，没能大规模商用。对模型进行剪枝与量化，可以有效的降低模型计算强度、参数大小和内存消耗，但往往带来巨大的精度损失。如何让算法真正落地成了亟待解决的问题。模型压缩作为软件方法，应用成本低，而且与硬件加速方法不矛盾，可以相互加成，细分有剪枝（Pruning）、量化（Quantization）、低秩分解（Low-rank factorization）、知识蒸馏（Knowledge distillation）。而量化和剪枝更适合训练后优化，即不需要从头开始训练模型。

4.2 模型量化的发展

模型量化目前主要有两个究方向，一个研究方向是权值共享，基本思想是多个网络连接的权重共用一个权值，还有一个是权值精简，既权值的低比特表示。

4.2.1 权值共享

权值共享最有代表性的研究工作就是由韩松大佬团队提出的聚类方式共享，论文对每一层 weight 矩阵利用 K-means 聚类算法聚类成若干个 cluster，用每个 cluster 的聚类中心值代表该 cluster 的权重，由于同一 cluster 的 weight 共享一个权重大小，因此我们只需要存储权值的 cluster 的 index 即可，通过查表获取该 index 对应的 value，为了减少精度损失，再通过训练微调的方式对权重进行补偿，所有的梯度信息按照权重矩阵之前的分组进行。下图很形象的说明了聚类量化与训练过程。

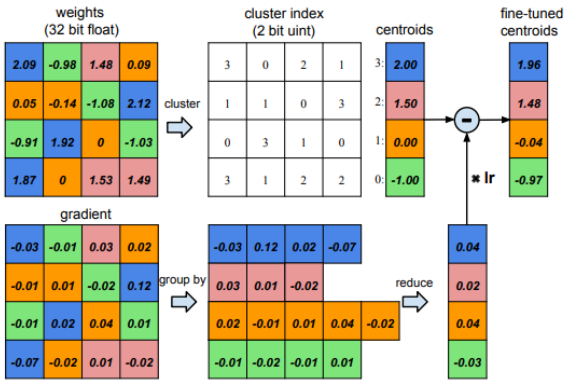


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

权值共享的另一个策略是哈希方式，W Chen 等人设计了一种新型网络架构 hashnet，利用哈希函数随机将网络连接权重分组到哈希桶（hash bucket），每个哈希桶内的网络连接共享相同的权重参数，该方法与特征哈希类似（feature hashing），将高维数据降到低维空间，该方法可显著减小模型体积，并对输出精度影响较小。

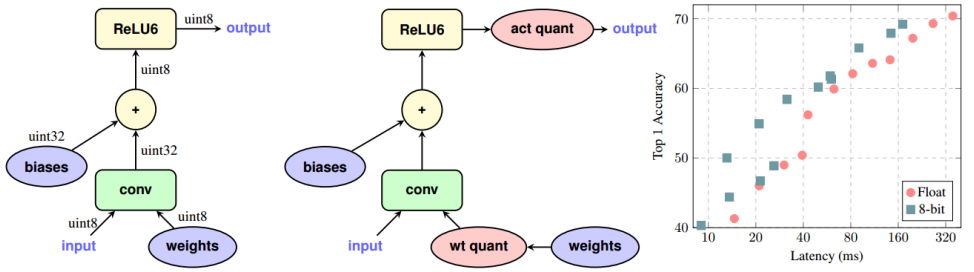
4.2.2 权值精简

我们知道，现在神经网络的 weight 和 bias 都是用单精度 4 字节的 float32 或者双精度 8 字节的 float64 的表示，为了降低模型的存储空间而达到模型压缩加速的目的，越来越多的学者企图通过更少的 bit 位来表示模型实际的浮点型权值。

权值精简的极限就是二值化，也就是 BinaryNet，该方法在 2016 年发表在 NIPS 的文章 Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1 中，论文提出了利用降低权重和输出的精度的方法来加速模型，因为这样会大幅的降低网络的内存大小和访问次数，并用 bit-wise operator 代替 arithmetic operator。这种方法的原理，在训练 BNN 时，将权重和输出置为 1 或 -1，可以直接将大于等于零的参数置为 1，小于 0 的置为 -1；或是将绝对值大于 1 的参数置为 1，将绝对值小于 1 的参数根据距离 ±1 的远近按概率随机置为 ±1。相应的还有 n-bit 精简和三值化网络（Ternary Network）等。

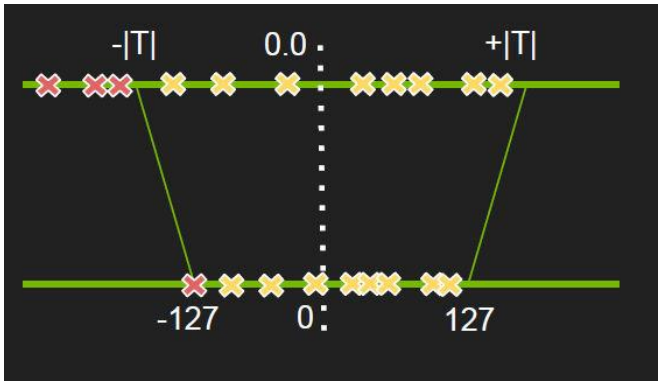
但现在业界有所使用的是 8 比特的权值精简，这也得益于一些芯片如 tensorRT、高通等对 8 比特量化做了硬件的适配。

主要解决方案有两个，一个是谷歌在 Quantization and training of neural networks for efficient integer-arithmetic-only inference 中提出的仅整数算术推理的量化方法，将 weights 和 input 量化位 uint8，bias 量化位 unit32，并将 unit32 的 accumulators 量化位 unit8，这里采用非线性量化，通过缩放系数 S(scale)和零点 Z(zero-point)确定实际值 r 对应的量化值 q，即 ，权重和输入的 S 和 Z 根据每一层的权值范围确定：scale = (max - min) / (qmax - qmin)，zero_point = qmin – min / scale，偏置的 S=S1*S2(S1、S2 分别为权值和输入的 scale)，Z=0。同时作者协同设计了一个模拟量化训练框架来保证端到端的精度。训练期间，反向和普通一样，用浮点数保存权值和偏置，前向的时候采用伪量化，反向的误差修正不量化，且 BN 层融入了 conv 层一起做量化，节省了单独做 BN 的计算，过程如下图所示。



(a) Integer-arithmetic-only inference (b) Training with simulated quantization (c) ImageNet latency-vs-accuracy tradeoff

另一个是英伟达的方案，考虑最简单的 min-max 映射，当正负分布不均匀的时候，是有一部分是空缺的，选择了找出一个合适的阈值，将无关的范围外细节给去掉，从而获得性能上的好处。当数据分布不均匀的时候，就把原始信息在映射前截断一部分，然后构成对称且分布良好的截断信息，再把这个信息映射到 int8 上去，从而减少动态范围资源的浪费，也减少精度损失。通过使用 KL 散度，比较量化前后两个分布的差异程度，选出阈值 T。当分布相同时，相对熵为零，当随机分布的差别增大时，相对熵也随即增大。



4.3 模型剪枝

剪枝是模型压缩的一个子领域，依据剪枝粒度可以分为非结构化、结构化剪枝，早期的一些方法是基于非结构化的，它裁剪的粒度为单个神经元。如果对 kernel 进行非结构化剪枝，则得到的 kernel 是稀疏的，即中间有很多元素为 0 的矩阵。除非下层的硬件和计算库对其有比较好的支持，pruning 后版本很难获得实质的性能提升。这几年的研究很多是集中在结构化剪枝上。依据实现方法可以大致分为基于度量标准、基于重建误差、基于稀疏训练的剪枝。

剪枝其实不是一个全新的问题，对神经网络的剪枝在上世纪八九十年代就有研究了，如上世纪 90 年代初当时经典的论文《Optimal brain damage》与《Second order derivatives for network pruning: Optimal Brain Surgeon》分别提出 OBD 和 OBS 方法，它们基于损失函数相对于权重的二阶导数（对权重向量来说即 Hessian 矩阵）来衡量网络中权重的重要程度，然后对其进行裁剪。

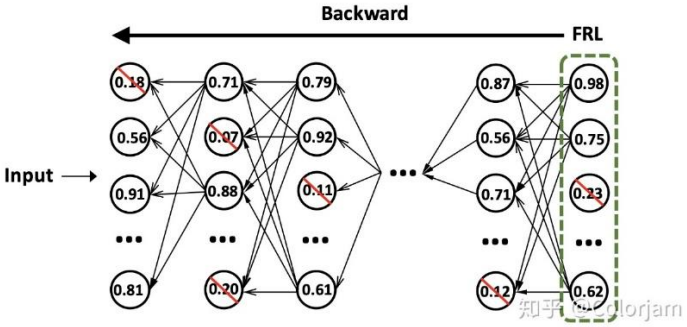
4.3.1 基于度量标准的剪枝

这类方法通常提出一个判断神经元是否重要的度量标准，依据标准计算出衡量神经元重要性的值，将不重要的神经元剪掉。在神经网络中可以用于度量的值主要分为 3 大块：Weight / Activation / Gradient。

如 2016 年经典论文《Pruning Filters for Efficient ConvNets》中把权重的绝对值作为衡量其重要性的手段。但训练出来的权重如果不稀疏不利于 pruning 怎么办，常用的办法是在训练时 loss 中加 regularizer，尤其是 L1 regularizer，从而使得权重稀疏化。Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration(CVPR2019) 把绝对重要性拉到相对层面，使用 Geometric Median 衡量 filter 的相对重要性。

一般来说，像 Relu 这样的激活函数会倾向产生稀疏的 activation；而权重相对而言不太容易是稀疏的。从这种意义上说，activation 更适合 pruning。如 2016 年论文《Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures》中提出采用 Average Percentage of Zeros，即 APoZ 来衡量 activation 的重要性。它定义为 activation 中为 0 的比例。从梯度角度出发的剪枝可追溯到上世纪 90 年代，代表性的是 Pruning Convolutional Neural Networks for Resource Efficient(ICLR2017)，对 activation 在 0 点泰勒展开。

4.3.2 基于重建误差的剪枝



这类方法通过最小化特征输出的重建误差来确定哪些 filters 要进行剪裁，即找到当前层对后面的网络层输出没啥影响的信息。ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression 采用贪心法，Channel Pruning for Accelerating Very Deep Neural

Networks(ICLR2017) 则采用 Lasso regression。NISP: Pruning Networks using Neuron Importance Score Propagation(CVPR2018) 通过最小化网络倒数第二层的重建误差，并将反向传播的误差累积考虑在内，来决定前面哪些 filters 需要裁剪。

4.3.3 基于稀疏训练的剪枝

这类方法采用训练的方式，结合各种 regularizer 来让网络的权重变得稀疏，于是可以将接近于 0 的值剪掉。Learning Structured Sparsity in Deep Neural Networks 用 group Lasso 进行结构化稀疏，包括 filters, channels, filter shapes, depth。由于每个通道的输出都会经过 BN，可以巧妙地直接稀疏 BN 的 scaling factor，比如 Learning Efficient Convolutional Networks through Network Slimming(ICCV2017) 采用了 L1 regularizer。

4.4 现阶段的趋势

随着 AutoML 的大潮，越来越多的东西开始自动化。如模型量化也有一些空间可以自动化，如 2018 年论文《HAQ: Hardware-Aware Automated Quantization》考虑网络中不同层信息的冗余程度不一样，因此可以用不同位数进行量化。

这几年机器学习最火热的分支之一 GAN，也在不断渗透到已有领域，在剪枝中也开始有它的身影。如 2019 年论文《Towards Optimal Structured CNN Pruning via Generative Adversarial Learning》采用了 GAN 的思想，让 generator 生成裁剪后网络，discrimintor 来判别是否属于原网络还是裁剪后网络，从而进行更有效的网络结构化裁剪。

剪枝方法与 NAS 结合的很好，如 Approximated Oracle Filter Pruning for Destructive CNN Width Optimization(ICML2019)平行操作网络的所有层，用二分搜索确定每层的剪枝数。Fine-Grained Neural Architecture Search 把 NAS 的粒度降到了通道，包含了空的操作即剪枝。

此外，论文 Once for All: Train One Network and Specialize it for Efficient Deployment 还提出将模型训练从网络结构设计中解耦出来，根据给定的场景，从训练一次的网络中搜索，选择出模型大小、结构最合适的子网络，该论文设计了一种训练方法，获得不同的结构设置下直接开发的网络，从而不重新训练的情况下支持不同的深度、宽度、核大小和分辨率设置。和传统的 Prune 比，他们的是在多个维度上 shrink，以及他们会 fine-tine 多个 sub-nets

参考文献

[1] Cheng Y, Wang D, Zhou P, et al. A survey of model compression and acceleration for deep neural networks[J]. arXiv preprint arXiv:1710.09282. 2017.

[2] LeCun Y, Denker J S, Solla S A. Optimal brain damage.In.Advances in neural information processing systems[J]. 1990: 598-605.

[3] B. Hassibi, D.G. Stork.Second Order Derivatives for Network Pruning: Optimal Brain Surgeon[C]. Advances in Neural Information Processing Systems.1993. 5: 164-171.

[4] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning[J], trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.2015.

[5] Lebedev V, Ganin Y, Rakhuba M, et al. Speeding-up convolutional neural networks using fine-tuned cp-decomposition[J]. arXiv preprint arXiv:1412.6553.2014.

[6] Ba J, Caruana R. Do deep nets really need to be deep?[C]. Advances in neural information processing systems. 2014: 2654-2662.

[7] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531. 2015.