DOI:10.7544/issn1000-1239.2020.20200291
57(9): 1839-1853, 2020

TensorFlow Lite:端侧机器学习框架

李双峰
(Google TensorFlow 团队 北京 100190)
(shuangfeng@google.com)

TensorFlow Lite: On-Device Machine Learning Framework

Li Shuangfeng
(*Google TensorFlow Team*, *Beijing* 100190)

Abstract TensorFlow Lite (TFLite) is a lightweight, fast and cross-platform open source machine learning framework specifically designed for mobile and IoT. It's part of TensorFlow and supports multiple platforms such as Android, iOS, embedded Linux, and MCU etc. It greatly reduces the barrier for developers, accelerates the development of on-device machine learning (ODML), and makes ML run everywhere. This article introduces the trend, challenges and typical applications of ODML; the origin and system architecture of TFLite; best practices and tool chains suitable for ML beginners; and the roadmap of TFLite.

Key words machine learning; on-device machine learning (ODML); TensorFlow; TensorFlow Lite; TFLite; mobile; IoT

摘 要 TensorFlow Lite(TFLite)是一个轻量、快速、跨平台的专门针对移动和 IoT 场景的开源机器学习框架,是 TensorFlow 的一部分,支持安卓、iOS、嵌入式 Linux 以及 MCU 等多个平台部署.它大大降低开发者使用门槛,加速端侧机器学习的发展,推动机器学习无处不在.介绍了端侧机器学习的浪潮、挑战和典型应用;TFLite 的起源和系统架构;TFLite 的最佳实践,以及适合初学者的工具链;展望了未来的发展方向.

关键词 机器学习;端侧机器学习;TensorFlow;TensorFlow Lite;TFLite;移动;物联网

中图法分类号 TP391

TensorFlow Lite[1] (TFLite)是一个轻量、快速、跨平台的专门针对移动和 IoT 应用场景的机器学习框架,是开源机器学习平台 TensorFlow[2-4] (TF)的重要组成部分.它致力于"一次转换,随处部署",支持安卓、iOS、嵌入式 Linux 以及 MCU 等多种平台,降低开发者使用门槛,加速端侧机器学习(on-device machine learning, ODML)的发展,推动机器学习无处不在.本文介绍了工业界端侧机器学习的最新趋势以及 TFLite 如何加速其发展,包括:

1) 端侧机器学习的趋势、挑战和典型应用,以及 TFLite 的起源;

2) TFLite 的系统架构;

3) TFLite 的最佳实践以及适合初学者的工具链;

4) 未来的发展方向.

1 TensorFlow Lite 推动端侧机器学习的发展

1.1 机器学习的普及以及 TensorFlow 的发展

机器学习的发展改变着语音交流、机器翻译、健康医疗和城市交通等我们生活中的诸多领域,越来越多的企业、组织和个人尝试用机器学习解决业务或者生活中遇到的难题.开发者数量的大量增加也意味着需要降低技术门槛,让更多人参与进来.

在 2015 年底,Google 开源了端到端的机器学习

收稿日期:2020-04-23;**修回日期**:2020-06-30

开源框架 TensorFlow[2-4]:它既用于研究,也用于大规模生产领域;既支持大规模的模型训练,也支持各种环境的部署,包括服务器和移动端的部署;支持多种语言比如 Python,C++,Java,Swift 甚至 JavaScript. TensorFlow 提供了全面灵活的工具生态,帮助解决各种挑战性的问题.而近年来移动化浪潮和交互方式的改变,使得机器学习技术开发也在朝着轻量化的端侧发展,于是 TensorFlow 团队在 2017 年底开源了 TFLite[5],一个轻量、快速、兼容度高的专门针对移动应用场景的深度学习工具,降低了端侧深度学习技术的门槛.

1.2 端侧机器学习的机遇和挑战

伴随移动和 IoT 设备的普及,世界以超乎想象的方式连接在一起.如今已有超过 30 亿的智能手机用户[6],以及超过 70 亿的联网 IoT 设备[7](不包括手机、电脑等).手机成本不断降低,并且随着微控制器(mircrocontrollers, MCU)和微机电系统(micro-electro-mechanical systems, MEMS)的发展,高性能低功耗的芯片使得"万物"智能具有了可能性.从智能穿戴、智能家居到共享单车,从工业控制到车载设备,这些设备都有了智能化的基础.我们有时把这些设备统称为边缘设备(edge device).

将边缘设备的数据传输到云端处理很多时候不是最经济有效的方式,它带来了延迟,降低了复杂网络条件下的可靠性,也引起了更多的隐私顾虑,从而影响用户体验.用户对交互的需求越来越高,快速、及时的智能反应是消费者的普遍期待,以智能音箱为例,唤醒的响应速度是良好体验的基础.

我们把在边缘设备上的运行机器学习统称为端侧机器学习(ODML),它为万物智能互联带来了新的机遇:

1) 更快、更紧密的交互方式,因为模型在本地执行的延迟小.比如及时的语音唤醒、直播视频的实时图像分割或者辅助驾驶的目标检测,都需要在本地执行.

2) 在复杂网络环境下仍可提供可靠服务,比如在网络基础设施相对落后的国家或偏远地区,在复杂环境下(比如隧道中),带宽可能有限或无法联网.

3) 更好的保护隐私,因为在本地进行数据收集和处理,减少了数据上传.

然而实现端侧机器学习有很多挑战,因为边缘设备:

1) 算力有限,限制了模型的复杂度.

2) 内存有限,限制了模型的大小.

3) 电池有限,需要模型运算效率更好.比如对智能手表而言,省电非常关键.

通常,移动设备上耗电最多的是无线电,如果在本地执行,尤其是有 DSP 或 NPU 等硬件加速器的情况下,电池续航时间就更长.

4) 计算硬件生态碎片化严重,比如 CPU,GPU,DSP,NPU 等,如何将这些异构硬件真正利用起来是一大难题.这与云端不同,在云端,硬件加速器供应商通常非常集中,如 NVIDIA GPU 或 TPU.

1.3 TensorFlow Lite 的起源和相关工作

TFLite 是在边缘设备上运行 TensorFlow 模型推理的官方框架,它跨平台运行,包括 Android,iOS 以及基于 Linux 的 IoT 设备和微控制器.

TensorFlow 适用于云端的大型、大功率设备,以及本地的工作站设备.当边缘设备的需求增加时,我们尝试简化 TensorFlow 并在移动设备上运行,这就是 TF Mobile 项目:它是一个缩减版的 TensorFlow 变体,简化了算子集,也缩小了运行库(runtime).然而,我们始终难以大大缩小运行库;同时运行库的扩展性方面也存在问题,如何将其映射到移动环境中所用的各种异构加速器上困难重重.

TFMini 是 Google 内部用于计算机视觉场景的解决方案,它提供了一些工具(比如 TOCO 转换工具),压缩模型(比如删掉训练相关的不必要节点),进行算子融合并生成代码.它将模型嵌入到二进制文件中,这样我们就可以在设备上运行和部署模型.TFMini 针对移动设备做了很多优化,性能优秀,但在把模型嵌入到实际的二进制文件中时兼容性存在较大挑战,因此 TFMini 并没有成为通用的解决方案.

基于 TF Mobile 的经验,也继承了 TFMini 和内部其他类似项目的很多优秀工作,我们设计了 TFLite:

1) 更轻量.在 32 b 安卓平台下,核心运行时的库大小只有 100 KB 左右,加上支持基本的视觉模型(比如 InceptionV3 和 MobileNet)所需算子时,总共 300 KB 左右,而使用全套算子库时,只有 1 MB 左右.

2) 特别为各种端侧设备优化的算子库.

3) 能够利用各种硬件加速,比如 GPU,DSP 等.

TFLite 兼具性能和通用性,已经完全取代了 TF Mobile 和 TFMini,成为 TensorFlow 针对移动和 IoT 设备的官方框架,被广泛使用.

其他相关工作包括:CoreML[8]是 iOS 平台的解决方案,而 TFLite 强调其优秀的跨平台能力.

中国的开源端侧机器学习框架代表包括腾讯NCNN[9]、小米MACE[10]和阿里巴巴MNN[11]等,它们专注于移动推理平台,而TFLite是TensorFlow生态的一部分,支持从训练到多种平台部署[3](比如TFX,TF.js,TFLite),并提供完整的工具链(比如TensorBoard).

1.4 端侧机器学习应用的蓬勃发展

端侧机器学习在图像、文本和语音等方面都有非常广泛的想象空间.全球有超过40亿[12]的设备上部署着TFLite,这个数字还在不断增加当中.Google的大量产品部署着TFLite,比如Google Assistant,Google Photos等;国际巨头比如Uber,Airbnb等[12],以及国内的许多大公司,比如网易、爱奇艺和WPS等,都在使用TFLite.

一方面,端侧机器学习能解决的问题越来越多元化,这带来了应用的大量繁荣.前期在图像和视频方面广泛应用,比如Google Lens[13],Google Photos,Google Arts & Culture[14].最近,离线语音识别方面有很多突破,比如Google Assistant宣布了完全基于神经网络的移动端语音识别[15],效果和服务器端十分接近,服务器模型需要2 GB大小,而手机端只需要80 MB.端侧语音识别非常有挑战,它的进展代表着端侧机器学习时代的逐步到来.一方面依赖于算法的提高,另外一方面TFLite框架的高性能和模型优化工具也起到了很重要的作用.离线语音可以带来很好的用户体验,比如无需网络环境即可使用,无隐私顾虑.Google Pixel 4手机上发布了Live Caption[16],自动把视频和对话中的语言转化为文字,大大提高了有听力障碍人群的体验(accessibility).更进一步,端侧自然语言处理将会有巨大的前景.我们发布了基于MobileBERT模型的问题回答系统[17-18],速度非常快,在普通CPU上可实时响应,创造了无缝的用户体验.

另外一方面,模型越来越小,无处不再.Google Assistant的语音功能部署在非常多元的设备上,比如手机端、手表、车载和智能音箱上,全球超过10亿设备.更激动人心的前景发生在IoT领域,TFLite可以支持微控制器(microcontrollers, MCU)[19],而MCU是单一芯片的小型计算机,没有操作系统,只有内存,也许内存只有几十KB.很多设备上都有MCU,全球有超过一千五百亿的MCU.MCU低功耗、便宜、无处不在.TFLite发布了若干MCU上可运行的模型,比如识别若干关键词的语音识别模型和简单的姿态检测模型,模型大小都只有20 KB左右,我们基于此可构建很有意思的应用,比如更智能的小玩具:它在MCU上运行上述模型,呼叫设定的特定昵称时就会发出某个声音,而当拿起玩具做一些设定的动作时就会响应.当随处可见的物品都在MCU上部署机器学习的模型时,智能开始无处不在.

在中国,在移动应用方面,网易使用TFLite做OCR处理[20];爱奇艺使用TFLite来进行视频中的AR效果[21],而WPS用它来做一系列文字处理[22].在IoT方面,出门问问智能音箱使用TFLite来做热词唤醒[23](对于智能音箱而言,准确、实时、轻量化低功耗的唤醒非常关键),科沃斯扫地机器人使用TFLite在室内避开障碍物[24].另外,TFLite也非常适合工业物联智能设备的开发,因为它很好地支持如树莓派及其他基于Linux SoC的工业自动化系统.创新奇智应用TFLite开发智能质检一体机、智能读码机等产品,应用到服装厂质检等场景[25].

2 TensorFlow Lite系统架构

我们设计TFLite,目标是:

1) 轻量.缩小运行库和模型大小,减少内存消耗,适用于更多设备.

2) 高性能.针对移动和IoT设备深度优化,利用多种硬件加速机器学习,比如利用ARM CPU最新指令、GPU、DSP和NPU.

3) 跨平台、兼容度高.支持安卓、iOS、嵌入式Linux以及MCU等多种平台,支持多种"一次转换,随处部署".

4) 易用.与TensorFlow紧密集成,实现从训练到部署过程流畅,提供丰富的平台相关API,提供丰富的模型库、完整的实例和文档,以及丰富的工具链,降低开发者使用门槛.

5) 可扩展性.模块化,易定制,容易扩展到更多硬件支持,定制更多算子.

图1展示了TFLite的主要组成部分:

1) TFLite模型转换器[26](converter).TFLite自带一个转换器,它可以把TensorFlow计算图,比如SavedModel或GraphDef格式的TensorFlow

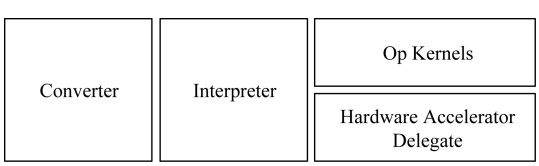

Fig.1 Main components of TFLite

图1 TFLite的主要组成部分

模型,转换成 TFLite 专用的模型文件格式,在此过程中会进行算子融合和模型优化,以压缩模型,提高性能.

2) TFLite 解释执行器(interpreter).进行模型推理的解释执行器,它可以在多种硬件平台上运行优化后的 TFLite 模型,同时提供了多语言的 API,方便使用.

3) 算子库(op kernels).TFLite 算子库目前有 130 个左右,它与 TensorFlow 的核心算子库略有不同,并做了移动设备相关的优化.

4) 硬件加速代理(hardware accelerator delegate).我们将 TFLite 硬件加速接口称 delegate(代理),它可以把模型的部分或全部委托给另一个硬件后台执行,比如 GPU 和 NPU.

图 2 展示了在 TensorFlow 2.0 中 TFLite 模型转换过程,用户在自己的工作台中使用 TensorFlow API 构造 TensorFlow 模型,然后使用 TFLite 模型转换器转换成 TFLite 文件格式(FlatBuffers 格式).在设备端,TFLite 解释器接受 TFLite 模型,调用不同的硬件加速器比如 GPU 进行执行.

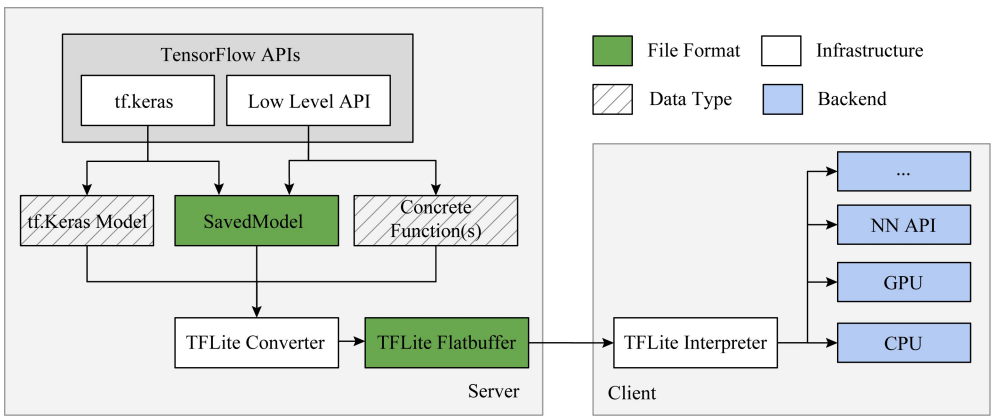

Fig. 2 TFLite model conversion

图 2 TFLite 模型转换过程

2.1 TensorFlow Lite 模型转换器

图 2 中描述了模型转换过程,转换器可以接受不同形式的模型,包括 Keras Model 和 SavedModel(TF 2.0 中推荐的格式),开发者可以用 tf.Keras 或者低层级的 TensorFlow API 来构造 TensorFlow 模型,然后使用 Python API 或者命令行的方式调用转换器(推荐使用 Python API,更灵活).比如:

1) Python API.调用 *tf.lite.TFLiteConverter*,可用 *TFLiteConverter.from_saved_model()*,或 *TFLiteConverter.from_keras_model()*;

2) 命令行:*tflite_convert --saved_model_dir=/tmp/mobilenet_saved_model --output_file=/tmp/mobilenet.tflite*

在 TF 1.x 版本中,还支持 GraphDef 格式,如果需要,请使用:*tf.compat.v1.lite.TFLiteConverter*.

转换器做了 2 类优化工作:

1) 算子优化和常见的编译优化,比如算子融合、常数折叠(constant folding)或无用代码删除等.TFLite 实现了一组优化的算子内核,转化成这些算子能在移动设备上实现性能大幅度提升.比如让我们将 Relu 融合到卷积等高级算子中,或优化 LSTM 算子.

2) 量化的原生支持.在模型转换过程中使用训练后量化(post-training quantization)非常简单,不需要改变模型,最少情况只需多加一行代码,设置 *converter.optimizations=*[tf.lite.Optimize.DEFAULT].

2.2 TensorFlow Lite FlatBuffers 格式

TFLite 模型文件格式采用 FlatBuffers[27].与 Protocol Buffers[28] 类似,FlatBuffers 是 Google 的一个开源的跨平台序列化格式,最初为视频游戏而设计.它在开发过程中更注重考虑实时性,内存高效,这在内存有限的移动环境中是极为关键的.它支持将文件映射到内存中(mmap),然后直接进行读取和解释,不需要额外解析.我们将其映射到干净的内存页上,减少了内存碎片化.另外,相对于 Protocol Buffers,它有更小的依赖,因此也会减小二进制文件大小.

TFLite 代码中 schema.fbs[29] 文件使用 FlatBuffers 定义了 TFLite 模型文件格式,我们摘取了其中一些关键样例代码,以帮助理解模型所包含的信息,如图 3 所示.TFLite 模型文件是一个层次的结构:

1) TFLite 模型(model)由子图(subgraph)构成,同时包括用到的算子库和共享的内存缓冲区.

2) 张量(Tensor,多维数组)用于存储模型权重,或者计算节点的输入和输出,它引用 Model 的内存缓冲区的一片区域,提高内存效率.

3) 每个算子实现有一个 OperatorCode,它可以是内置的算子,也可以是自定制算子,有一个名字.

4) 每个模型的计算节点(operator)包含用到的算子索引,以及输入输出用到的 Tensor 索引.

5) 每个子图包含一系列的计算节点、多个张量,以及子图本身的输入和输出.

```
table Model {
  operator_codes:[
    OperatorCode];
  subgraphs:[SubGraph];
  buffers:[Buffer];
}

table SubGraph {
  tensors:[Tensor];
  inputs:[int];
  outputs:[int];
  operators:[Operator];
}

table Tensor {
  shape:[int]
  type:TensorType
  buffer:uint
}
```

```
table OperatorCode {
  builtin_code:BuiltinOperator;
  custom_code:string;
}

table Operator {
  opcode_index:uint;
  inputs:[int];
  outputs:[int];
  builtin_options:BuiltinOptions;
  custom_options:[ubyte];
}
```

Fig. 3 TFLite schema.fbs samples[29]

图 3 TFLite schema.fbs 样例代码[29]

模型转换器最早来源于 TFMini 项目,称为 TOCO 转换器.最近我们基于 Google 最新的机器学习编译技术 MLIR[30] 重写了转换器,将算子的 TensorFlow dialect 映射到算子的 TFLite dialect.它提供了更好的转换错误追踪和调试功能,也支持了更多的新模型(比如 Mask R-CNN,Mobile BERT),特别是对控制流(control flow)有更好的支持.另外,也让 TFLite 转换器具有更好的可扩展性,同时可以利用机器学习编译技术的最新成果.

2.3 TensorFlow Lite 解释执行器

TFLite 解释执行器[31] 针对移动设备从头开始构建,具有 3 个特点:

1) 轻量级.在 32 b 安卓平台下,编译核心运行时得到的库大小只有 100 KB 左右,如果加上所有 TFLite 的标准算子,编译后得到的库大小是 1 MB 左右.它依赖的组件较少,力求实现不依赖任何其他组件.

2) 快速启动.既能够将模型直接映射到内存中,同时又有一个静态执行计划,在转换过程中,我们基本上可以提前直接映射出将要执行的节点序列.采取了简单的调度方式,算子之间没有并行执行,而算子内部可以多线程执行以提高效率.

3) 内存高效.在内存规划方面,采取了静态内存分配.当运行模型时,每个算子会执行 *prepare* 函数,它们做必要的内存分配.我们会分配一个单一的内存块,而这些张量会被整合到这个大的连续内存块中.不同张量之间甚至可以复用内存以减少内存分配.

参照图 3 中 TFLite 模型格式,我们会执行模型的子图,而根据数据依赖关系,子图中的计算节点已经被提前静态规划好,保证每个计算节点在执行前它所需要的张量已经被之前的计算准备好.所以,总体上依次执行即可.

使用解释执行器通常需要包含 4 步:

1) 加载模型.将 TFLite 模型加载到内存中,该内存包含模型的执行图.

2) 转换数据.模型的原始输入数据通常与所期望的输入数据格式不匹配.例如,可能需要调整图像大小,或更改图像格式,以兼容模型.

3) 运行模型推理.使用 TFLite API 执行模型推理.

4) 解释输出.解释输出模型推理结果.比如,模型可能只返回概率列表,而我们需要将概率映射到相关类别,并将其呈现给最终用户.

TFLite 提供了多种语言的 API,正式支持的有 Java,C++和 Python,实验性的包括 C,Object C,C#和 Swift.大家可以从头自己编译 TFLite,也可以利用已编译好的库,Android 开发者可以使用 JCenter Bintray 的 TFLite AAR,而 iOS 开发者可通过 CocoaPods 在 iOS 系统上获取.

图 4 是使用 Java API 的一个示例:创建输入

```
import org.tensorflow.lite.Interpreter;

// Allocate input and output.
ByteBuffer input=ByteBuffer.allocateDirect(inputSize);
ByteBuffer output=ByteBuffer.allocateDirect(outputSize);

// Omitted: Fill input buffer, customize options.
try (Interpreter tflite=new Interpreter("/tmp/awesome_model.
tflite", options)) {
  // Invoke the interpreter.tflite.
  run(input, output);
  // Omitted: display the output.
}
```

Fig. 4 TFLite Java API code samples

图 4 TFLite Java API 代码样例

缓冲区和输出缓冲区,填充输入,并定制推理选项(比如是否使用 GPU).之后创建 TFLite 执行器,输入 TFLite 模型并执行.

2.4 高性能算子库和可扩展性

目前 TFLite 约有 130 个算子(op kernels),大多同时支持浮点和量化类型,可以在这里找到所有支持的算子[32].很多算子专门针对 ARM CPU 进行了优化,包括基于汇编和 intrinsics 级别的 NEON 指令集的多种优化.针对矩阵相乘,算子之前浮点运算使用 Eigen,量化运算使用 gemmlowp[33],最近我们重新设计了一个叫 Ruy[34] 的高性能矩阵相乘库,统一了浮点运算和量化运算,并且性能有较大提升.特别针对一些重度使用的算子做了深度优化,比如涉及到卷积或者 LSTM 的算子.另外,不少算子支持多线程执行.

算子的定义基于 C 语言接口 *TfLiteRegistration*,包括 4 个函数(*init*,*free*,*prepare*,*invoke*),开发者可方便自定义算子.

在创建解释器时,用户可以提供自定义的算子解析器(OpResolver),从而控制模型执行所用到的算子实现,如图 5 所示.使用内置算子解析器(BuiltinOpResolver)时使用默认的算子.每个算子都有自己的版本控制.我们还提供机制,把不用的算子自动删掉.

```
TfLiteRegistration * Register_SIN() {
  //"SinPrepare" corresponds to prepare and invoke functions in
    the interface.
  static TfLiteRegistration r={nullptr, nullptr,
    SinPrepare, SinEval};
  return &r;
}
// Register customized op "Sin".
tflite::ops::builtin::BuiltinOpResolver builtins;
builtins.AddCustom("Sin", Register_SIN());
```

Fig. 5 TFLite customized op

图 5 TFLite 自定义算子

当遇到不能支持的算子时,可以自定义算子,另外一种选择是,复用 TensorFlow 算子,称为 Select TF Ops,只需要多加一行转换器参数就可以开启:

converter.target_spec.supported_ops=[*tf.lite.OpsSet.TFLITE_BUILTINS*, *tf.lite.OpsSet.SELECT_TF_OPS*]

它可能带来的问题是所依赖的运行库更大,同时 TensorFlow 算子在移动设备中未必足够优化.

2.5 CPU 性能

CPU 最具普适性,其性能最为关键,因此我们持续不断提升 CPU 性能.矩阵运算(GEMM[35])效率对于深度学习非常关键,过去我们使用 gemmlowp[33] 库进行量化矩阵乘法,使用 Eigen 库进行浮点乘法.最近,我们从头开发了一个针对移动环境 CPU 特别优化的矩阵乘法库 Ruy[34],统一了浮点运算和量化运算.之前面向桌面和云端的矩阵乘法库则更加专注于大矩阵乘法的峰值性能,而 Ruy 对各种矩阵大小的乘法都表现优异:不仅适合大矩阵,也适合当前 TFLite 应用中最常见的矩阵运算(往往是小矩阵或者多种多样的矩阵形状).Ruy 目前支持浮点数和 8 b 整型,主要针对 ARM CPU(64 b 和 32 b),而我们也在针对英特尔 x86 架构进行优化.

自 TensorFlow 1.15 版本开始,Ruy 在所有 ARM 设备上默认启用,大范围地加速了模型,特别是那些卷积比较多的视觉模型,将延迟降低了 1.2～5 倍,尤其在具有 NEON 点积内置函数的硬件上.图 6 以最常见的 MobileNet V1 为例,浮点模型和量化模型都有明显提高.

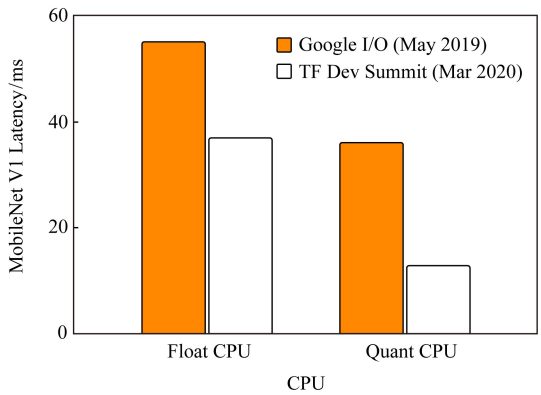

Fig. 6 Single thread CPU on Pixel4

图 6 Pixel 4 上单线程 CPU 性能

性能提升工作需要持续不断的努力,另一个即将到来的新突破,是全新的高度优化后的浮点卷积核库 XNNPACK[36].在 TFLite 支持的所有关键浮点卷积模型上的测试结果表明,XNNPACK 可以进一步提高执行速度.

以 MobileNet V1 浮点模型为例,如图 7 所示,在多种硬件平台上,使用 XNNPACK 后,单线程 CPU 性能提升达 20%～200%.比如在 X86 Windows 平台上,有了 2 倍的提升.我们将在 TensorFlow 2.3 版本中集成 XNNPACK,并计划在 2.4 版本中对于浮点模型,在所有平台中默认使用 XNNPACK.

2.6 硬件加速器代理

我们将 TFLite 硬件加速接口称 delegate[37](代理),它可以把模型计算图的部分或全部代理给

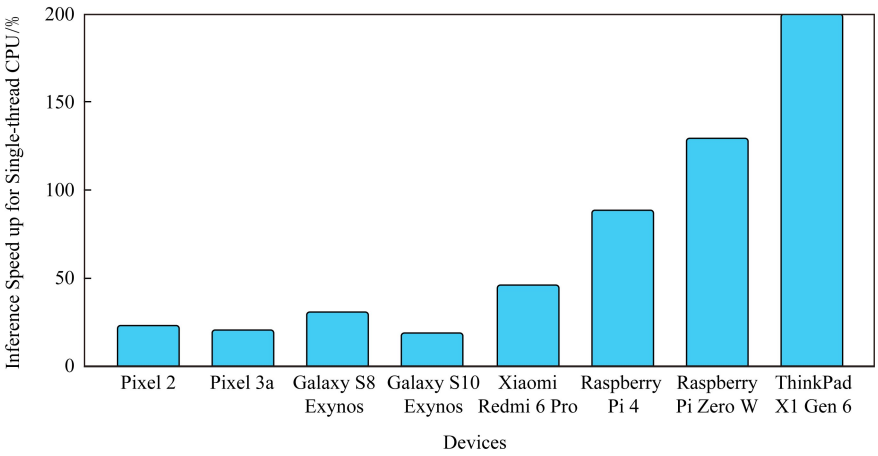

Fig. 7 Single thread CPU performance improvement with XNNPACK (Jun 2020)

图 7 使用 XNNPACK 后单线程 CPU 性能提升比例(2020 年 6 月)

另一个执行器,让它们在硬件加速器中执行.在图 8 示例中,整个计算图一部分在 CPU 中执行,另一部分子图被代理给硬件加速器执行.

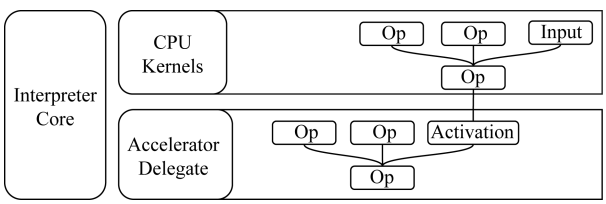

Fig. 8 TFLite hardware delegate example

图 8 TFLite 硬件加速器代理示意

我们不是把算子逐一放到加速器上执行,而是将计算图的整个子图在加速器上运行,这对于 GPU 或 NPU(神经网络加速器)等需要在设备上尽可能多地进行计算,且中间没有 CPU 互操作的设备而言,是一个巨大的优势.

TFLite 有着非常丰富的硬件加速器支持:在 Android 系统中,支持 NNAPI,GPU,EdgeTPU 和 Hexagon DSP delegates;在 iOS 系统中,有 Metal 和 CoreML delegates.

1) GPU delegate[38].适用于跨平台.GPU delegate 可在 Android 和 iOS 上使用,支持 32 b 和 16 b 浮点的模型,对许多浮点卷积模型实现了大幅度速度提升,尤其是较大的模型.不过,运行库大小会有小幅度的增加.GPU 的驱动可以有不同的后端,Android 系统有 OpenGL,OpenCL 和 VulKan 后端,以及 iOS 上基于 Metal 的后端.我们最近增加了对 OpenCL 的支持,在多个视觉模型上进行的测试表明,基于 OpenCL 的 GPU delegate 的性能提升为 CPU 上的 4~6 倍和 OpenGL 的 2 倍.图 9 是三者在 Pixel 4 上测试后端性能的对照结果.

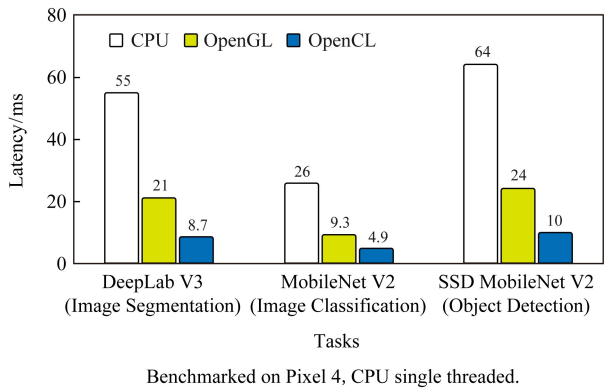

Fig. 9 TFLite GPU performance

图 9 TFLite GPU 性能

2) Android NNAPI delegate[39].适用于较新的 Android 设备.NNAPI delegate 可用于在具有 GPU,DSP 和 NPU 的 Android 设备上加速模型.它在 Android 8.1(API 27+)或更高版本中可用.NNAPI[40]是 Android 系统中用于加速机器学习的抽象层.在 NNAPI 和 TFLite 中,我们会发现高级算子的定义有很多相似之处,因为二者是紧密联系在一起进行开发的.NNAPI 是 Android 平台级的硬件加速器抽象层,我们可以通过 TFLite 调用它,而供应商比如高通则为 NNAPI 提供 DSP 或 GPU 的驱动程序.在 Android Q 上,NNAPI 真正进入了一个良好的稳定状态,功能和算子方面都正在接近 TFLite 的同等水平.有越来越多的用户和硬件供应商予以采用,这促进了这些驱动程序的发展.

3) Hexagon delegate[41](DSP).适用于较旧的

Android 设备.Hexagon DSP 是一种微处理器,常见于大量使用高通骁龙 SoC 的安卓手机.与 CPU 相比,新的 TFLite Hexagon delegate 利用 DSP 实现了 MobileNet 和 InceptionV3 等模型,性能大幅提升,提升幅度可达 3～25 倍,同时 CPU 和 GPU 的能效也得到了提升[42].它可以在不完全支持 NNAPI 的旧版 Android 设备上使用.

4) Core ML delegate[43].适用于较新的 iPhone 和 iPad.CoreML 是在苹果设备上使用的机器学习框架,它还提供了在 Neural Engine 上运行机器学习模型的 API.CoreML delegate 允许在 CoreML 和 Neural Engine(如果设备有相关芯片)上运行 TFLite 模型,以更低的功耗实现更快的推理速度.在含有 Neural Engine 的 iPhone XS 以及后续发布的设备上,测试表明各种计算机视觉模型的性能提高了 1.3～11 倍[44].

图 10 给出了使用 TFLite GPU delegate 的 Java 示例,只需要多加几行代码,非常简单,而其他 delegate 也类似.值得注意的是,图中的 delegate.bindGlBufferToTensor 可以使用 OpenGL 纹理作为计算图输入,这样数据就不用复制,减少在 CPU 和 GPU 之间来回传送数据.

```
GpuDelegate delegate=new GpuDelegate();
Interpreter.Options=new Interpreter.Options().addDelegate
(delegate);
try (Interpreter tflite=new Interpreter("/tmp/awesome_model.
tflite", options)) {
  delegate.bindGlBufferToTensor(tflite.getInputTensor(0),
    glBufferHandle);
  tflite.run(inputs, outputs);
}
```

Fig. 10 Use TFLite GPU delegate in Java

图 10 使用 TFLite GPU delegate 的 Java 示例

可扩展性是 TFLite 很重要的设计目标,如果一些硬件后端默认还不支持,完全可以自己定制 delegate,只需要:

1) 定义一个新的 delegate 算子,它负责计算被代理的子图.

2) 创建一个 *TfLiteDelegate* 实例,它负责注册新定义的 delegate 算子,并可以把计算图中的一些子图替换为新的 delegate 算子.

这样,应用开发者可以利用新的加速器,同时硬件厂商也可以扩展对 TFLite 的支持,从而触及更多的用户.

我们正在持续改进如何更方便地实现一个新 TFLite delegate,以及提供可扩展的工具链来验证测试这一新 delegate.目前推荐扩展 *SimpleDelegate* 相关 APIs[45]来实现,具体可以参考我们提供的简单案例[46]来进一步了解,包括如何集成到我们的测试和验证工具链当中.更多请参考 delegate 使用开发指南[37].

2.7 TensorFlow Lite 微控制器版

1.4 节提到,MCU 没有操作系统,只有极小内存,低功耗、便宜、无处不在.真正在 MCU 上部署机器学习模型变得很普及时,智能开始无处不在.TFLite 可以支持微控制器 MCU,为 IoT 领域的智能化带来很多想象空间.

TFLite 微控制器版(TFLite for Microcontrollers)[47]接受和 TFLite 同样的 FlatBuffers 模型格式,让同样的模型可运行在手机和 MCU 上,减轻了开发者负担.它的核心运行库在 Arm Cortex M3 上小于 16 KB,加上一些关键词检测的算子,总共只需要 22 KB.

它支持 ARM Cortex-M 系列芯片以及其他架构比如 ESP32.也可以作为 Arduino 库,对 MBed 开发环境也有很好的支持.最近,Cadence 宣布旗下的 Tensilica HiFi DSP 系列也支持 TFLite 微控制器版[48].

MCU 上的模型都非常小,低功耗,只能完成简单的功能,不过可以常驻内存,一直处于等待触发状态.一般常见的用法是级联触发:比如先用 MCU 的一个模型检测是否有声音,如果是,另一个 MCU 模型判断这是不是人的声音(排除外界噪音),更进一步才开启设备 CPU 上更强大的语音识别功能.如果设备需要开启 CPU 持续监听外界的声音,功耗会很大.

我们开源了很多有意思的样例,比如:

1) 识别若干关键词的语音识别模型,只有 20 KB 左右.

2) 魔法棒.只需要 20 KB 左右的模型就可以做姿态检测,这很适合一些使用加速计数据的应用.

3) 人物识别.250 KB 的视觉模型识别摄像头是否有人出现.

3 使用 TensorFlow Lite 的最佳实践

3.1 解决模型转换挑战

目前 TFLite 约有 130 个算子,大多同时支持浮点和量化类型,可以在这里找到所有支持的算子[32].

TensorFlow 有一些语义在 TFLite 中尚未得到很好的原生支持,同时在模型转换过程中算子进行了优化和融合.总结一下 TensorFlow 算子和 TFLite 算子的兼容性:

1) 大多 TFLite 算子支持 float32 和量化(int8, uint8),不过很多算子尚不支持 bfloat16 或 string.

2) TFLite 目前只支持 TensorFlow 的某些固定格式,而广播只支持有限的一些 ops(比如 *tf.add*,*tf.mul*,*tf.sub*,*and tf.div*,*tf.min*,*tf.max*).

3) 一些 TensorFlow 算子有严格对应的 TFLite 算子,比如 *tf.nn.avg_pool*,*tf.nn.conv2d*,*tf.nn.depthwise_conv2d*,*tf.nn.l2_normalize*,*tf.nn.max_pool*,*tf.nn.softmax*,*tf.nn.top_k* 等,以及 *tf.matmul*,*tf.one_hot*,*tf.reduce_mean*,*tf.reshape*,*tf.sigmoid*,*tf.squeeze*,*tf.strided_slice*,*tf.transpose* 等.详细清单请参考文献[32].

4) 另外很多 TensorFlow 算子,没有一一对应的 TFLite 算子,不过 TFLite 仍可以支持,在模型转换过程得到优化.这些 TensorFlow 算子可能被删除(比如 *tf.identity*),替换(比如 *tf.placeholder* 被换成张量),或者融合为更复杂的操作(比如 *tf.nn.bias_add*).详细清单请参考文献[32].

如遇到模型转换问题,请尝试如下解决方法:

1) 自定义算子.可以自定义新的算子,或者提供定制的更优化的算子.这种方式对于开发者而言最灵活,可控性高.

2) 使用其他等价算子.可以替换为其他 TFLite 能支持的算子,假设模型更改后精度和速度等方面没有明显损失.

3) Select TF Ops.可以重用 TensorFlow 算子,缺点是二进制文件大小会有所增加,速度也可能会慢些.

4) RNN/LSTM 的支持.在 TF 1.x 版本中我们提供了一种叫做 OpHint 的机制[49],开发者需要使用 TFLite 封装的 RNN/LSTM API,让可以标注 TensorFlow 模型的一些计算节点,直接转成 TFLite 中优化的 RNN/LTSM 算子.TF 2.x 中不再支持 OpHint,我们提供了基于 Keras 的更简单灵活的支持[50-51],不过需要模型使用 Keras 构建.

5) 控制流(control flow)支持.最新的基于 MLIR 的模型转换器已经对此有不错的支持.

6) 动态张量形状(dynamic tensor shape)的支持.目前对动态张量形状支持有限,需要在转化时将输入设置成固定大小,随后在运行时调用 ResizeInput.设置输入为固定大小的技巧包括调整图片大小,或者为文字加填充.

一方面,算子在不断增加中,我们也在为 TFLite 增加动态张量形状等更多新功能的支持.持续关注社区和 github 更新,可尝试每日更新的代码是否已支持;还可以向社区发 issue 或者直接贡献代码.

3.2 进行模型优化

压缩模型可以加速模型执行,更好地利用硬件加速器,特别是当一些加速器只能接受全整型的模型时.模型压缩领域是很热的研究话题,而我们则更关注足够简单的开发体验,希望用户只需要几行代码就可以实现.TensorFlow 提供了一个模型优化工具包(TensorFlow model optimization toolkit, MOT)[52],简化模型优化过程.MOT 被很好地集成到了 TFLite 工具链中,可在 TFLite 模型转换时轻松启用.

MOT 目前主要支持 2 类优化:

1) 量化(quantization).降低模型参数精度,比如将浮点转换成整型.

2) 剪枝(pruning).减少模型参数,比如去掉那些不重要的权重(比如为 0 的权重).

从开发者使用体验来看,又可以分为:

1) 训练后(post-training).比如训练后量化(post-training quantization),不需要改变模型,非常简单.

2) 训练中(training-aware).需要改动模型,在训练的时候优化.

推荐训练后量化(post-training quantization)[53],足够简单,只要能满足需求.

Table 1 TFLite Post-Training Quantization

表 1 TFLite 训练后量化

Techniques	Benefits	Hardware
Dynamic Range Quantization	4x smaller, 2x-3x speedup	CPU
Full Integer Quantization	4x smaller, 3x+ speedup	CPU, DSP, EdgeTPU
Float16 Quantization	2x smaller, GPU acceleration	CPU/GPU

最初引入的是动态范围量化(dynamic range quantization),它可以动态地对输入和激活函数做量化和去量化,是一种混合量化:当有量化算子支持时,调用量化算子核,否则使用浮点算子核,整个执行过程混合了 2 类算子.精度上会有所损失,损失多大取决于具体模型.图 11 展示了如何将 32 b 浮点数映射到 8 b 整数.它贵在简单,只需在转化时多加一行

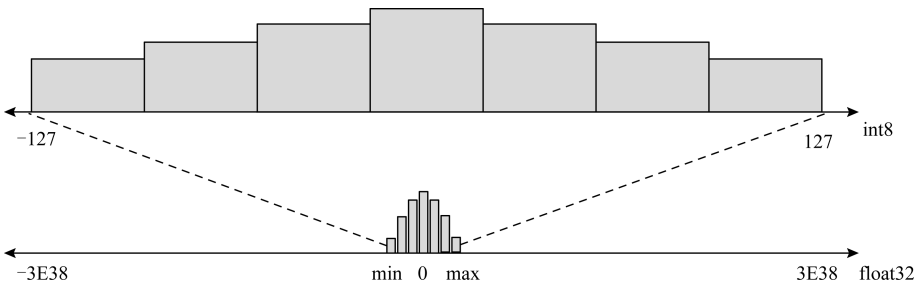

Fig. 11 Quantization: map 32 b float to 8 b integer

图 11 量化:将 32 b 浮点数映射到 8 b 整数

代码:*converter.optimizations* = [tf.lite.Optimize.DEFAULT].

而全整型量化[54]可以同时量化权重和激活函数,这样所有的权重和计算都是整形的,可以进一步改善延迟,减少峰值内存使用量,以及利用仅支持整数的硬件加速器.

为进一步提高精度,我们支持了按轴量化或按通道量化:不会对整个张量使用同一组量化参数,而是对张量中每个通道使用不同的量化参数.为此,需要使用代表性数据集来评估激活函数和输入的动态范围,帮助确定量化参数,这样可以实现与训练时量化大体相当的精确度.对于一个基于图片的模型而言,或许只需要输入 30 张图片,就能够探索出量化和输出值的空间.

开发者只需创建一个输入数据生成器,并将代表性数据提供给 TFLite 转换器即可:

converter.representative_dataset = *representative_dataset_gen*

如有 GPU,则考虑使用 16 b 浮点数,进一步减少精度损失:

converter.target_spec.supported_types = [tf.lite.constants.FLOAT16]

训练时优化,需要对于模型训练过程做一些改变,相对更复杂.为尽量简化用户体验,MOT 针对 Keras 提供了非常便捷的 API.目前可以支持:

1) 训练时量化(training-aware quantization).可以量化整个 Keras 模型和部分 Keras 模型节点.

2) 剪枝(pruning).去掉一些权重,让模型更稀疏.

比如量化整个 Keras 模型只需要增加一行:

quantized_model = *mot.quantization.keras.quantize_model*(*keras_model*)

MOT 在压缩 Google 的一些关键模型发挥着重要作用,比如 1.4 节提到的离线语音模型.我们一直致力于开发一些更易用的工具,既简化操作,又能够在模型压缩比和加速性能方面同样优异.比如,之前的训练时量化工具需要在模型中插入伪量化(fake quantization)节点,相对复杂,因此我们发布了新的更易使用的基于 Keras 训练时量化[55].

我们也在探索更多压缩方法的支持,比如张量压缩算法或者模型蒸馏.

3.3 减少运行库大小

除了压缩模型本身,我们也希望减少运行库大小.一个技巧就是只链接所需要的算子,因为通常一个模型只用到部分算子,我们称之为选择性注册(selective registration).

TFLite 有很好的可扩展性,可以自己定义算子和算子解析器.我们在/tensorflow/lite/tools/下提供了工具,给定一个模型,可以扫描用到的算子,自动生成一个注册实际算子的代码文件,这样利用自定义的算子解析器,就可以删除不用的算子内核.

3.4 选择正确的模型

选择合适的模型非常关键,这很大程度上取决于应用的需求,比如:

1) 模型大小的限制;

2) 模型延迟的需求;

3) 精度要求;

4) 硬件运行环境.

可以尝试不同的模型,对各个方面做取舍.比如,如果精度要求高而硬件许可,可以选择较复杂的模型;而速度最关键时,选择简单而精度低的模型.

研究领域发布的模型日新月异,开发人员很难时刻跟进,另外,论文上的精度更多是理想情况,是否在真实场景下有稳定表现还需要根据实际场景评估.通常情况下,我们可以从一些经典的针对移动特别优化的在工业界久经考验的模型开始,比如 MobileNet.

可以尝试在目标手机上运行一些 TFLite 的参考应用[56],感受模型实际取得的效果,比如对象追踪应用的流畅程度.

TFHub 上提供了一系列的经典模型[57],既有预训练的 TF 模块,可做迁移学习后转换到 TFLite 模型,也可以直接下载 TFLite 模型.机器学习是一个飞速发展的领域,也许每隔几个月就有新的模型刷新记录.我们在努力确保最前沿(SOTA)的模型可以在 TFLite 上运行.

最近,我们增加了对 EfficientNet-Lite(图像分类模型系列)[58],MobileBERT[59] 和 ALBERT-Lite[60](支持多种 NLP 任务的轻量级版本 BERT)的支持.

EfficientNet-Lite 是一种新颖的图像分类模型,可通过减少计算和参数的数量级来实现 SOTA 的准确性.它针对 TFLite 量化方式进行了优化,在损失较低精度(几乎可忽略)的同时大大提升了推理速度,并可以运行在 CPU,GPU 和 EdgeTPU 上[61].

图 12 中,在相似的高精度下,EifficientNet-Lite4 比 Inception V4 有显著的速度提升.

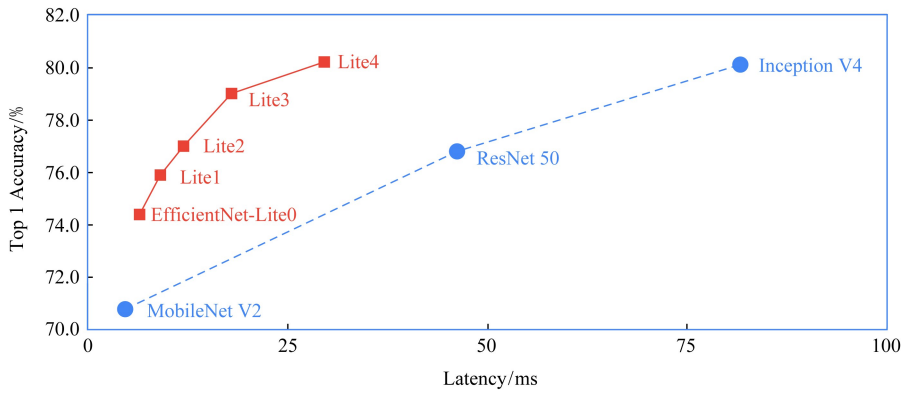

Fig. 12 Benchmarked on Pixel 4 CPU with 4 threads (March 2020)

图 12 在四线程 Pixel 4 CPU 上做基准测试(2020 年 3 月)

MobileBERT[62] 和 ALBERT-Lite[63] 是流行的 BERT 模型的优化版本,该模型在一系列 NLP 任务(包括问答、自然语言推断等)上均达到了 SOTA 的准确度.图 13 中,MobileBERT 的体积是 BERT 的 1/4,速度是 BERT 的 4 倍,同时保持了相近的准确度.ALBERT-Lite 的体积甚至更小,仅有 BERT 的 1/6,也就是 MobileBERT 体积的 2/3,在速度上 ALBERT-Lite 稍逊于 MobileBERT.我们还在探索量化版本的 MobileBERT,它是 BERT 的 1/16,速度是 BERT 的 8 倍,同时也保持了相近的准确度,

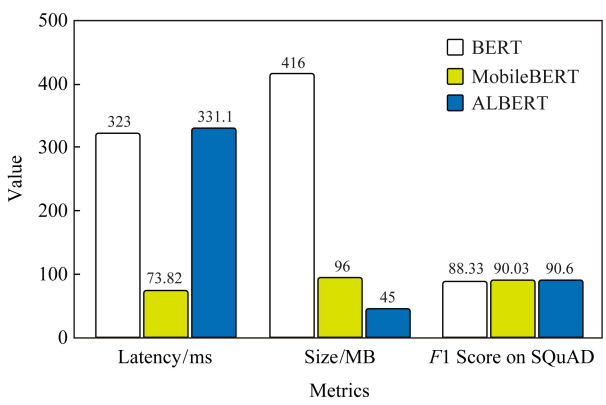

Fig. 13 Benchmarked on Pixel4, Float32 QA model, 4 threads CPU

图 13 在 Pixel 4,Float32 问答模型,4 线程 CPU 上评测

MLPerf 社区正尝试基于此建立移动硬件加速的 NLP 测试基准.

3.5 性能的最佳实践

TFLite 性能测试工具[64]可以找出在特定设备上运行时的性能瓶颈,输出耗时长的算子,还可以插入不同的计算后端,并探索它究竟是如何影响推理性能的.它支持对内部事件(如算子调用)的检测日志记录,可以通过 Android 的系统跟踪(system tracing)[65]进行追踪.

以下是性能分析工具给出的分析示例,及性能提升的备选解决方案:

1) 如果可用的 CPU 内核数量少于推理线程的数量,则 CPU 调度开销可能会导致性能下降.可以在应用程序中重新调度其他大量占用 CPU 的任务,以避免与模型推断重叠,或尝试调整解释器线程的数量.

2) 如果算子没有完全被代理到 GPU 上执行,那么模型计算图中的某些部分将在 CPU 上执行,而不是按预想的在硬件加速器上执行.可以将不支持的算子替换为相似的已支持算子.

尽可能优化模型,比如量化.尽可能地使用硬件加速器,也注意一些性能负担,比如内存零拷贝.

当然,移动平台可能具有多种硬件加速器,而模型的选择也很多(比如模型类型、量化或浮点),组合

起来比较复杂.同时这些加速器也可能需要被用作其他用途,比如 GPU 可能需要预留给视频处理,这带来了优化的复杂度.通常情况下,需要在不同平台上尝试运行应用,统计数据,根据数据进行优化.我们也在探索提供更好的开发者工具支持.

3.6 TensorFlow Lite 的使用限制

由于 TensorFlow 算子丰富,而新模型也不断出现,虽然 TFLite 覆盖了很多常见的模型,对于普通开发者,模型转换仍是较大的痛点.请参考 3.1 节部分解决,比如自定义算子,利用 Select TF Ops.

TFLite 目前对动态张量形状支持有限,需要在转化时将输入设置成固定大小,随后在运行时调用 ResizeInput.很多 TFLite ops 可能还不支持 bfloat 和 string.另外,TFLite 目前只支持 TensorFlow 的某些固定格式,而广播只支持有限的一些 ops(比如 $tf.add$, $tf.mul$, $tf.sub$, and $tf.div$, $tf.min$, $tf.max$)[32].

硬件加速部分,delegate 的算子不支持仍是主要挑战.总体而言,CPU 算子有较好的覆盖率,而 GPU 和 DSP 等覆盖率相对小一些;而 NNAPI 的算子则更新速度较慢,且其硬件驱动的性能不一,建议进行实际模型性能测试.如碰到问题,欢迎到社区里反馈.

社区对于个性化的应用和联邦学习(federated learning[66])的兴趣逐步增强,因此对于端侧训练的需求越来越多.目前 TFLite 对于端侧训练还不支持.我们提供了一个 TFLite 案例,允许有限的端侧模型迁移,从而让端侧模型个性化,不过使用相对复杂[67].

对于 RNN/LSTM,在 TF 1.x 版本中需要使用 TFLite 封装的 RNN/LSTM API[49],为模型节点加上相关标注,相对复杂.而在 TF 2.x 中灵活度较高,需要模型使用 Keras 构建[50-51].

4 适合初学者的工具

4.1 预训练模型和完整参考示例

我们提供了预训练模型的代码库和实现这些模型的示例应用[56,68],开发者无需编写任何代码即可在实际设备上试用 TFLite 模型,并可以在 github 找到完整应用代码,包括模型的前处理和后处理.包括多种案例,比如对象追踪、风格迁移和问题回答,也包括不同平台的例子,比如 Android、WiOS、树莓派及 MCU.

比如,在手机上实现问题问答[17]是一个很有挑战的问题,我们发布了基于 MobileBERT 的参考应用,学术界的多个 BERT 模型可以在这个应用上运行.风格迁移模型[69]启发了 Google Arts & Culture 的产品新特性[14].

对于视频处理应用,MediaPipe[70]提供了很好的框架,可以和 TFLite 配合使用.MediaPipe 也有丰富应用例子,比如手势追踪.

另外,社区 github 项目"Awesome TFLite"[71],收集了很多有意思的示例.

4.2 TensorFlow Lite Model Maker

TFLite Model Maker[72]使用迁移学习,可让开发者在自己的数据集上应用最前沿的机器学习模型.它将复杂的机器学习概念封装在直观的 API 中,无需机器学习专业知识,只需几行代码即可训练最新的图像分类模型:

```
data = ImageClassifierDataLoader.from_folder('flower_photos/')
model = image_classifier.create(data)
loss, accuracy = model.evaluate()
model.export('flower_classifier.tflite')
```

Fig. 14 TFLite Model Maker example

图 14 TFLite Model Maker 示例

Model Maker 支持 TensorFlow Hub 上的许多最新模型,如 3.5 节 EfficientNet-Lite.如果想获得更高的准确率,仅需修改一行代码,即可切换到不同的模型架构.目前支持图片分类和文本分类任务,我们还在不断扩展其功能.

4.3 TensorFlow Lite Support 库

执行模型推理前,通常需要对输入数据做转换,称为前处理,比如图片裁剪和大小挑战.而推理之后,如何解读数据,又需要做后处理.

为简化开发流程,我们提供了 TFLite Support[73]库,它提供了一系列工具库.目前支持图片处理库,更多数据类型(比如 NLP)和更多平台(比如 iOS)的支持正在进展当中.

4.4 模型元数据、自动代码生成和 Android Studio 工具

直接使用 TFLite 解释器,它的输入输出都是张量.这带来了 2 个问题:

1)TFLite 模型的使用者将需要确切地知道一个 1×224×224×3 张量的含义.比如它是位图吗?特别是模型创建团队和使用团队不是同一个时,更容易出问题.

2)对高维数据进行转换时容易出错,例如把 Bitmap 转换为 RGB 浮点数组或者 ByteArray.

我们希望把模型调用并做前后处理的整个复杂

过程都封装到一个简单的 API 中,开发者只需要调用简单 4~5 行代码,而且可以直接使用他们熟悉的原生数据,比如 Android 开发时直接用 Bitmap,而不是转化为 ByteArray.

为此,TFLite 增加了对模型元数据的支持[74],这让模型创建者可以使用类型化的对象来描述模型的输入和输出,方便模型共享和自动化处理.更进一步,提供了一个 Android 代码生成器[75],给定模型,它可以读取元数据,自动生成封装好的 Java 类,它可以自动调整图片大小,对其进行归一化且从 ByteArray 进行转换.

更进一步,我们正将此功能集成到 Android Studio 的 ML model binding 工具中[76],只需要把模型导入到 Android Studio 中就可以生成代码,目前已提供试用版,请参考文献[77].

5 未来的方向

TFLite 作为开源项目,为了更好地与社区互动,我们提前公开了开发计划[78],以确保所从事的工作和优先级清晰明了.

性能是我们持续关注的重点.我们正在集成 XNNPACK 以便于进一步优化浮点模型,还在提升训练后量化的性能,以便加快 CPU 推理速度.增加更多优化的算子,并让现有的硬件加速器 delegate 支持更多算子,以便加速更多模型,支持更多的硬件加速器.在有多个异构加速器的情况下,探索如何提供更简易的工具,帮助发挥出其最佳性能.

简化开发者体验很关键.我们会持续不断更新最前沿的端侧模型以及相关的示例,展示更多的可能性.增强 TFLite Model Maker,支持更多任务,例如目标检测,支持 BERT 系列的 NLP 任务.更丰富的 TFLite Support 库.扩展模型元数据和代码生成工具,以支持更多用例,以及与 Android Studio 的无缝集成,进一步简化体验.提供更便捷工具,帮助用户根据模型来减少算子库,压缩运行时间.

进一步让 TensorFlow 模型到 TFLite 模型转化更流畅,比如更好地支持 RNN/LSTM 和动态形状、模型转换过程更好的语义匹配(比如只转换部分计算图).长期看来,如何更好地融合 TensorFlow 和 TFLite,共享算子和核心库,同时,也兼具移动平台的优化.

提供更丰富的模型优化工具,比如剪枝、张量压缩和蒸馏技术.

在 TFLite 微控制器版中,和社区合作支持更多芯片平台、更多算子,同时提供更多适合 MCU 的样例模型和应用.

更好地支持联邦学习,保护用户隐私.更好地支持端侧训练,支持个性化应用.

致谢 感谢 TensorFlow 团队,特别是 TensorFlow Lite 团队的杰出工作,设计并开源 TensorFlow Lite,加速端侧机器学习的发展.感谢 Google 相关团队持续对 TensorFlow Lite 的贡献和反馈.感谢广大硬件平台的支持.感谢最广大开发者对 TensorFlow 社区的贡献,持续推动其发展.

参 考 文 献

[1] TensorFlow. TensorFlow Lite: ML for mobile and IoT [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/

[2] Abadi M, Barham P, Chen Jianmin, et al. TensorFlow: A system for large-scale machine learning[C]//Proc of the 12th USENIX Symp on Operating Systems Design and Implementation (OSDI 2016). Berkeley, CA: USENIX Association, 2016: 265-283

[3] TensorFlow. TensorFlow: An end-to-end open source machine learning platform [EB/OL]. [2020-04-22]. https://www.tensorflow.org

[4] TensorFlow. TensorFlow open-source project in github [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow

[5] TensorFlow. TensorFlow Lite open-source project in github [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite

[6] Dea S O. Number of smartphone users worldwide from 2016 to 2021 [EB/OL]. [2020-04-22]. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[7] Lueth K L. State of the IoT 2018: Number of IoT devices now at 7B-Market accelerating [EB/OL]. [2020-04-22]. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[8] Apple. CoreML: Integrate machine learning models into your app [EB/OL]. [2020-04-22]. https://developer.apple.com/documentation/coreml

[9] Tencent. Tencent NCNN: High-performance neural network inference framework optimized for the mobile platform [CP/OL]. [2020-04-22]. https://github.com/Tencent/ncnn

[10] Xiaomi. Xiaomi MACE: A deep learning inference framework optimized for mobile heterogeneous computing platforms [CP/OL]. [2020-04-22]. https://github.com/XiaoMi/mace

[11] Alibaba. Alibaba MNN: A lightweight deep neural network inference engine [EB/OL]. [2020-04-22]. https://github.com/alibaba/MNN

[12] Davis T, Alumbaugh T J. TensorFlow Lite: ML for mobile and IoT devices (TF Dev Summit'20)[R/OL]. [2020-03-13]. https://www.youtube.com/watch?v=27Zx-4GOQA8

[13] Fang Boya. On-Device computer vision in Google Lens [R/OL]. [2020-04-01]. https://www.youtube.com/watch?v=IbHWbT2Q1dc

[14] Luo M. Transform your photo in the style of an iconic artist [EB/OL]. [2020-04-02]. https://blog.google/outreach-initiatives/arts-culture/transform-your-photo-style-iconic-artist/

[15] SchalkwykJ. An all-neural on-device speech recognizer[EB/OL]. [2019-03-12]. https://ai.googleblog.com/2019/03/an-all-neural-on-device-speech.html

[16] Kemler B. If it has audio, now it can have captions [EB/OL]. [2019-10-16]. https://www.blog.google/products/android/live-caption/

[17] TensorFlow. TensorFlow Lite: Question and answer [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/models/bert_qa/overview

[18] TensorFlow. MobileBERT demo in TF World 2019 Keynote [R/OL]. [2019-11-01] https://youtu.be/zxd3Q2gdArY?t=2950(from 49′10″)

[19] Wikipedia. Microcontroller [EB/OL]. [2020-04-22]. https://en.wikipedia.org/wiki/Microcontroller

[20] Lin Huijie. Netease Youdao: AI leader in online education in China (TF Dev Summit'19)[R/OL]. [2020-04-22]. https://www.youtube.com/watch?v=Y8Nfcjg0faw

[21] SmileAR Engineering Team at iQIYI. SmileAR: iQIYI'smobile AR solution based on TensorFlow Lite [EB/OL]. [2020-07-12]. https://blog.tensorflow.org/2019/07/smilear-iqiyis-mobile-ar-solution-tensorflow-lite.html

[22] Xiong Longfei, Du Cheng, Chen Ronghua, et al. KingSoft WPS: Document image dewarping based on TensorFlow [EB/OL]. [2019-12-10]. https://blog.tensorflow.org/2019/12/kingsoft-wps-document-image-dewarping.html

[23] Mobvoi. Mobvoi: Use TensorFlow Lite for hot word detection [EB/OL]. [2018-09-11]. https://www.sohu.com/a/253161006_100262248

[24] ECOVACS. ECOVACS: AI based room cleaner [EB/OL]. [2020-04-22]. https://xw.qq.com/cmsid/20190416A05E9I00

[25] AInnovation. AInnovation: Computer vision for industrial quality control [EB/OL]. [2020-04-22]. http://www.geekpark.net/news/242839

[26] TensorFlow. TensorFlow Lite converter [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/convert

[27] Google. FlatBuffers: An efficient cross platform serialization library [EB/OL]. [2020-04-22]. https://google.github.io/flatbuffers/

[28] Google. Protocol buffers: A language-neutral, platform-neutral extensible mechanism for serializing structured data [EB/OL]. [2020-04-22]. https://developers.google.com/protocol-buffers

[29] TensorFlow. TensorFlow Lite schema file [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/schema/schema.fbs

[30] MLIR. MLIR: Multi-Level IR compiler framework [EB/OL]. [2020-04-22]. https://mlir.llvm.org/

[31] TensorFlow. TensorFlow Lite inference [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/guide/inference

[32] TensorFlow. TensorFlow Lite and TensorFlow operator compatibility [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/guide/ops_compatibility

[33] Jacob B. gemmlowp: A small self-contained low-precision GEMM library [EB/OL]. [2020-04-22]. https://github.com/google/gemmlowp

[34] Jacob B. The ruy matrix multiplication library [EB/OL]. [2020-04-22]. https://github.com/google/ruy

[35] Warden P. Why GEMM is at the heart of deep learning [EB/OL]. [2015-04-20]. https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

[36] Dukhan M. XNNPACK: High-efficiency floating-point neural network inference operators for mobile, server, and Web [EB/OL]. [2020-04-22]. https://github.com/google/XNNPACK

[37] TensorFlow. TensorFlow Lite delegates [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/delegates

[38] TensorFlow. TensorFlow Lite GPU delegate [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/gpu

[39] TensorFlow. TensorFlow Lite NNAPI delegate [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/nnapi

[40] Android. Android neural networks API [EB/OL]. [2020-04-22]. https://developer.android.com/ndk/guides/neuralnetworks

[41] TensorFlow. TensorFlow Lite hexagon delegate [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/hexagon_delegate

[42] Nosseir K, Joglekar S. Accelerating TensorFlow Lite on Qualcomm hexagon DSPs [EB/OL]. [2019-12-16]. https://blog.tensorflow.org/2019/12/accelerating-tensorflow-lite-on-qualcomm.html

[43] TensorFlow. TensorFlow Lite CoreML delegate [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/coreml_delegate

[44] Jeong T, Nosseir K. TensorFlow Lite Core ML delegate enables faster inference on iPhones and iPads [EB/OL]. [2020-04-02]. https://blog.tensorflow.org/2020/04/tensorflow-lite-core-ml-delegate-faster-inference-iphones-ipads.html

[45] TensorFlow. TFLite simple delegate APIs [CP/OL]. [2020-06-28]. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/delegates/utils/simple_delegate.h

[46] TensorFlow. TFLite simple delegate examples [CP/OL]. [2020-06-28]. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/delegates/utils/dummy_delegate

[47] TensorFlow. TensorFlow Lite for Microcontrollers [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/microcontrollers

[48] Cadence.Cadence Tensilica HiFi IP accelerates AI deployment with support for TensorFlow Lite for microcontrollers[EB/OL]. [2020-03-11]. https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2020/cadence-tensilica-hifi-ip-accelerates-ai-deployment-with-support.html

[49] TensorFlow. TensorFlow Lite: Convert RNN models[EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/convert/rnn

[50] TensorFlow. TFLite LSTM support for Keras [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/examples/experimental_new_converter/Keras_LSTM_fusion_Codelab.ipynb

[51] TensorFlow Lite User Group. TFLite fused LSTM/RNN ops conversion support [EB/OL]. [2020-06-28]. https://groups.google.com/a/tensorflow.org/g/tflite/c/Ub4apUvblN8/m/mtEizC2CAwAJ

[52] TensorFlow. TensorFlow model optimization toolkit [EB/OL]. [2020-04-22]. https://www.tensorflow.org/model_optimization

[53] TensorFlow. TensorFlow Lite: Post training quantization [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/performance/post_training_quantization

[54] TensorFlow Model Optimization Team. TensorFlow model optimization toolkit—Post-Training integer quantization[EB/OL]. [2019-06-11]. https://blog.tensorflow.org/2019/06/tensorflow-integer-quantization.html

[55] TensorFlow Model Optimization Team. Quantization aware training with TensorFlow model optimization toolkit—Performance with accuracy [EB/OL]. [2020-04-08]. https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html

[56] TensorFlow. TensorFlow Lite models and reference applications [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/models

[57] TFHub. TFLite models on TFHub [EB/OL]. [2020-04-22]. https://tfhub.dev/s?deployment-format=lite

[58] TFHub. EfficientNet-Lite on TFHub [EB/OL]. [2020-04-22]. https://tfhub.dev/s?deployment-format=lite&q=efficientnet%20lite

[59] TFHub. MobileBERT on TFHub [EB/OL]. [2020-04-22]. https://tfhub.dev/tensorflow/mobilebert/1

[60] TFHub. ALBERT-Lite on TFHub [EB/OL]. [2020-04-22]. https://tfhub.dev/tensorflow/albert_lite_base/1

[61] Liu Renjie. Higher accuracy on vision models with EfficientNet-Lite [EB/OL]. [2020-03-16]. https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html

[62] Sun Zhiqing, Yu Hongkun, Song Xiaodan, et al. MobileBERT: A compact task-agnostic BERT for resource-limited devices (ACL 2020)[C/OL]. [2020-04-22]. https://arxiv.org/abs/2004.02984

[63] Lan Zhenzhong, Chen Mingda, Goodman S, et al. ALBERT: A Lite BERT forself-supervised learning of language representations [C/OL]. [2020-04-22]. https://arxiv.org/abs/1909.11942

[64] TensorFlow. TFLite model benchmark tool [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark/

[65] Android. Android system tracing [EB/OL]. [2020-04-22]. https://developer.android.com/topic/performance/tracing

[66] McMahan B, Ramage D. Federated learning: Collaborative machine learning without centralized training data [EB/OL]. [2017-04-06]. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[67] Senchanka P. Example on-device model personalization with TensorFlow Lite [EB/OL]. [2020-04-22]. https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html

[68] TensorFlow. TensorFlow Lite models and reference applications [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/examples

[69] TensorFlow. Artistic style transfer with TensorFlow Lite [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/models/style_transfer/overview

[70] Google. MediaPipe: A cross-platform framework for building multimodal applied machine learning pipelines [CP/OL]. [2020-04-22]. https://github.com/google/mediapipe

[71] Reid M M. Awesome TFLite [EB/OL]. [2020-04-22]. https://github.com/margaretmz/awesome-tflite

[72] TensorFlow. TFLite model maker [CP/OL]. [2020-04-22]. https://github.com/tensorflow/examples/tree/master/tensorflow_examples/lite/model_maker

[73] TensorFlow. TensorFlow Lite Android support library [CP/OL]. [2020-04-22]. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/experimental/support/java

[74] TensorFlow. TFLite Metadata [EB/OL]. [2020-04-22]. https://www.tensorflow.org/lite/convert/metadata

[75] TensorFlow. TFLite Android Codegen[2020-04-22]. https://www.tensorflow.org/lite/guide/codegen

[76] Android. Android studio ML model binding [EB/OL]. [2020-04-22]. https://developer.android.com/studio/preview/features#tensor-flow-lite-models

[77] Lam H. New tools for finding, training, and using custom machine learning models on Android [EB/OL]. [2020-06-28]. https://android-developers.googleblog.com/2020/06/tools-for-custom-ML-models.html

[78] TensorFlow. TensorFlow Lite Roadmap [EB/OL]. [2020-04-20]. https://www.tensorflow.org/lite/guide/roadmap

Li Shuangfeng, born in 1982. MSc. His main work area is machine learning framework. In the past, he worked on large-scale search systems and geo products etc.