



NVIDIA 新架构 GPU 为机器学习应用带来的性能提升的研究 与评估

毕业设计答辩

刘子汉

10152130243@stu.ecnu.edu.cn

华东师范大学
计算机科学与软件工程学院
计算机科学与技术系

2019.05.15



大纲

大纲

简介

背景

相关工作

实验平台

实验内容

Benchmark

矩阵乘加

矩阵乘法

卷积

CUDA

卷积神经网络 (cuDNN)

支持向量机 (SMO-SVM)

TensorRT

Tensor Flow

总结

展望





简介

- 2017Q3, NVIDIA 发布新架构 GPU Tesla V100 及其中的张量核心, 且宣称矩阵乘加性能提升达 9.3 倍。
- Stefano 等人的研究中, 相同情况下其新架构 GPU 性能提升幅度仅有 4-6 倍, 如下图所示。

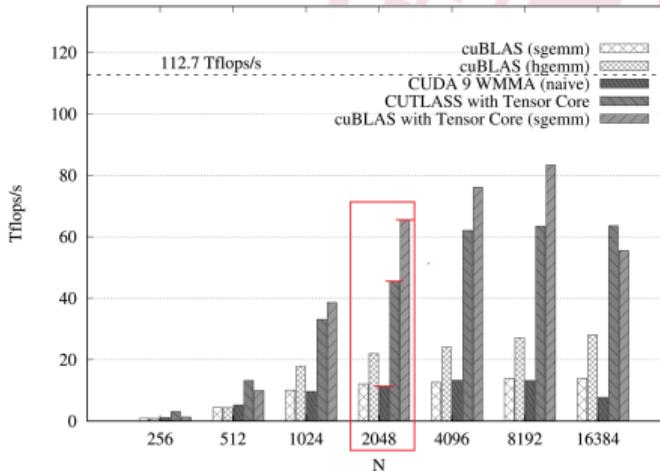
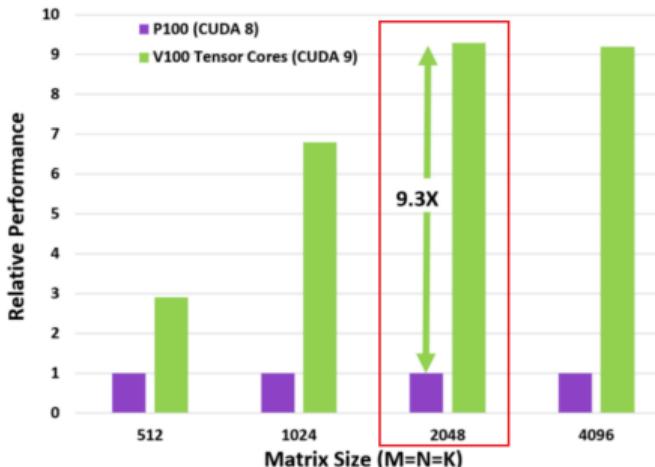


图: 官方白皮书性能与实际研究中性能比较



简介

- 在实际使用框架如 Tensor Flow 搭建的模型中提升幅度更低。在特定结构的网络中开启张量核心仅能带来 60%-80% 的提升。
- 本文将从 Python 源码、CUDA C 源码、PTX 中间代码、SASS 硬件代码的层面，借助卷积神经网络和支持向量机这两种经典的应用，对新架构 GPU 为机器学习应用带来的性能提升进行评估，尝试在代码层面进行优化，并提出设想。
- 具体评估的应用遵循自底向上的结构：
 - Benchmark 样例 (矩阵乘法、**矩阵乘加**、卷积运算)
 - 基于 CUDA 源码的应用 (卷积神经网络、支持向量机)
 - 基于 Tensor Flow 的应用 (卷积神经网络)
- 除训练过程外，最后使用 TensorRT 以及 Jetson 对部署、推理过程进行优化。



背景

- 机器学习与 GPU：目前绝大部分机器学习应用都需要 GPU 进行加速，而 NVIDIA GPU 长期占据高性能计算的市场。
- NVIDIA GPU 结构：自上而下分为图形处理器簇 (GPC)、纹理处理器簇 (TPC)、流多处理器 (SM)。流多处理器中有若干种处理单元如整数、浮点、逻辑单元。
- 伏特架构/图灵架构：在流处理器中加入了张量核心的新架构，分别对应计算能力 7.0 与 7.5，图灵是消费级芯片，屏蔽了一些硬件。
- 张量核心：专为矩阵乘加设计的硬件，以半精度浮点进行运算 (FP16)，以 wmma 指令批量执行原先整数点积指令与累加指令执行的任务。
- 纹理内存：访问时将二维空间上的周围数据加载进入缓存，其余存储系统为加载一行。
- 线程束：内含 32 个 GPU 线程，作为基本的调度、同步单元。
- 矩阵乘加：通用矩阵乘法，即两矩阵相乘再与偏置累加。大量存在于神经网络计算。
- TensorRT 与 Jetson：TensorRT 是一个 GPU 推理引擎，用于优化训练完毕的模型，加速推理。Jetson 是 NVIDIA 开发的面向嵌入式应用的芯片。



相关工作

- GPGPU-SIM: PTX 中间代码执行的软件层面模拟。
- SMart, PerfSIM: SASS 硬件代码执行的软件层面模拟以及 RTL 仿真。
- ThunderSVM: 并行支持向量机。
- Leng J.: 大型集群的性能、能耗优化。
- Mahmoud K.: 访存优化
- ? 张量核心



实验平台

表: 实验平台

项目	内容
CPU	AMD Ryzen ThreadRipper 2990WX 32C64T @ 3.0GHz
主板	MSI X399
内存	CORSAIR DDR4 3200 @ 16-15-15-34-1T 128GB
GPU	NVIDIA Geforce RTX 2080TI (Turing)
硬盘	INTEL750 NVMe PCIe 1.2TB * 2 @ RAID 0
系统	Windows 10 64-bit build 17763
CUDA	Ver. 10.1, 10.0, 9.2, 9.0
CUTLASS	Ver. 1.2, 1.3
其他	Jetson TX2 *



Benchmark:: 矩阵乘加

由于新老架构 GPU 在参数、外围设备等方面均有改进，为了重点研究张量核心的性能，本文中的实验均在 RTX 2080TI 上通过开启/关闭张量核心进行评估。

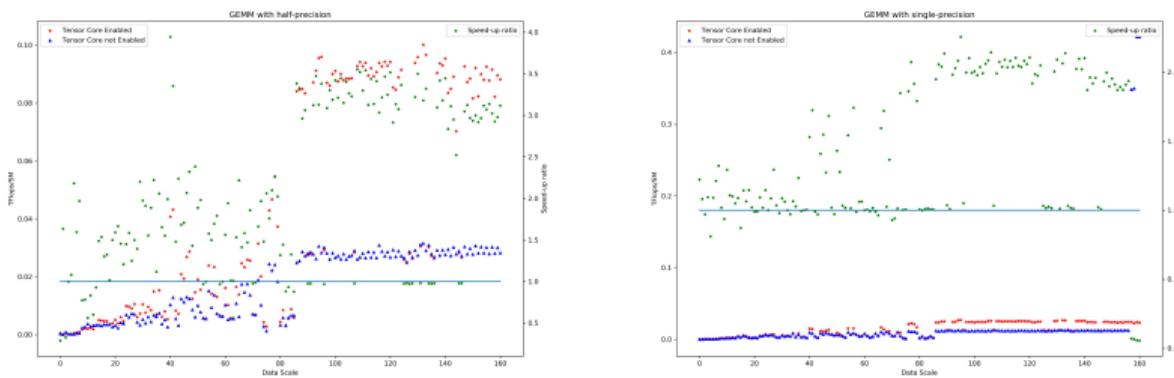


图: 不同计算量下开启和关闭张量核心的性能 (半精度/单精度)

在计算量较大的情况下，开启张量核心后半精度性能提升 3-4 倍，单精度性能提升 2 倍。



Benchmark:: 矩阵乘加

使用 nvprof 和 NSight 进行分析：

表：开启/关闭张量核心的对比

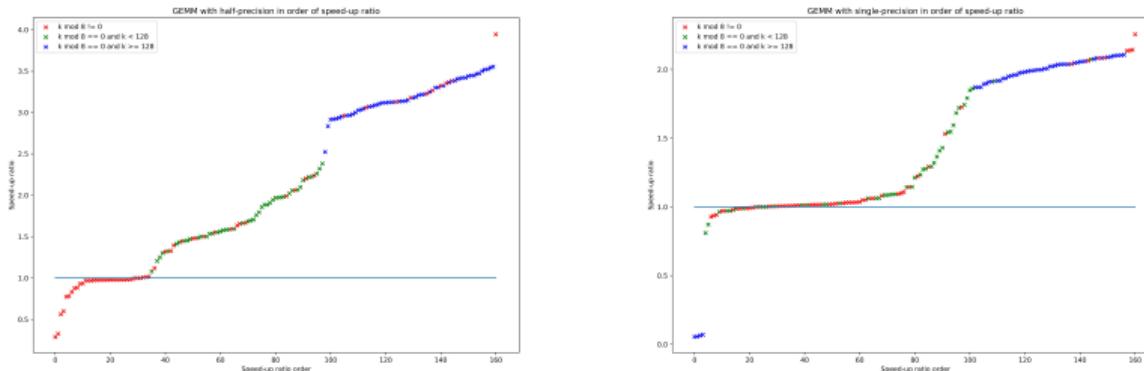
项目	开启张量核心	关闭张量核心
CUDA 设备同步耗时	186.15s	543.51
CUDA 设备同步耗时占比	79.29%	91.40%
一次乘加所需计算指令	一条 wmma	若干条 idp/idp4a+ 累加指令
每条计算指令延迟	wmma: 8 时钟周期	idp/idp4a: 4 时钟周期
上下文切换时间占比	44.39%	52.52%

开启张量核心后设备同步、上下文切换等死时间减少，原因为张量核心整合多次计算为一次。并行可扩展性强 (更大规模的 MMA) (**进一步扩大矩阵乘加指令的规模**)。结果中可见设备同步耗时仍然很长，目前 CUDA 仅支持线程束级别的同步，导致频繁的同步 (更细粒度的线程同步机制)。



Benchmark:: 矩阵乘加

根据官方文档说明，张量核心对于矩阵裁切形状较为敏感，故将实验结果按加速比排序并按形状特征着色，形状特征分为：能够被 32 整除、能够被 8 整除，无法被整除。



图：不同计算量下开启和关闭张量核心的性能 (半精度/单精度)-按加速比排序并着色

可见，当矩阵三维中 K 维度为 32 的整数倍时性能提升幅度最大，实际使用时，应对其数据规模、超参数进行优化以确保最大提升幅度。



Benchmark:: 矩阵乘法

在使用矩阵乘法评估 GPU 性能前，业内采用矩阵乘法对 GPU 在机器学习中的性能进行评估。本文考察使用 CUDA 中 cuBLAS 提供的 API，以及不使用 API 自行实现矩阵乘法的方法下矩阵乘法的性能。可见自行实现矩阵乘法性能极低，应尽量使用 API 完成基本的计算。

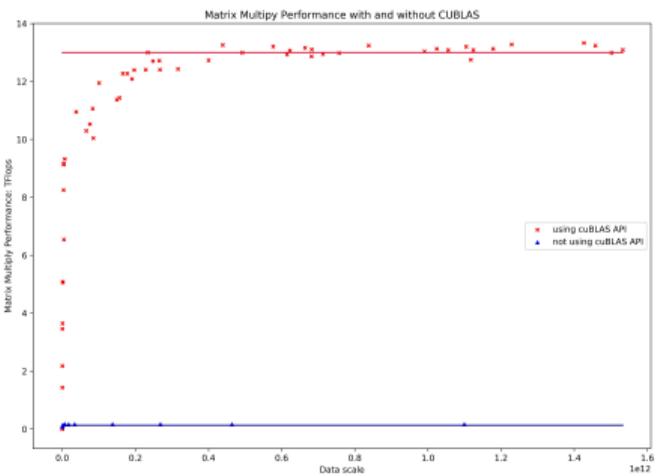


图: 使用和不使用 cuBLAS 库的单精度矩阵乘法性能



Benchmark:: 卷积

卷积大量存在于神经网络、图像处理计算中，本节评估如下几种卷积计算方式，这些方式分别具有各自特点。

表: 卷积计算方式及特征

卷积计算方式	描述
基于 FFT 的 CUDA 内建卷积	使用 CUDA 提供的基于 FFT 的卷积库
基于 FFT 的自定义卷积	使用 CUDA 提供的 FFT 库实现卷积
基于矩阵乘加的 CUDA 内建卷积	使用 CUDA 提供的利用张量核心的 API
直接卷积 (全局内存)	直接计算每一个单元的值，基于全局内存
直接卷积 (纹理内存)	直接计算每一个单元的值，基于纹理内存



Benchmark:: 卷积

结果中大尺寸图像与小尺寸图像差异较大。

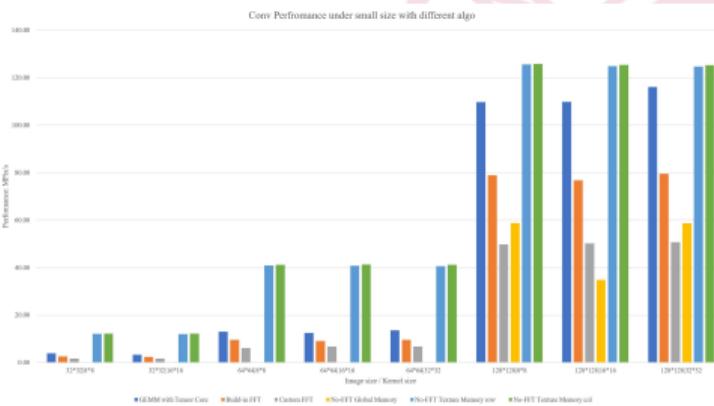
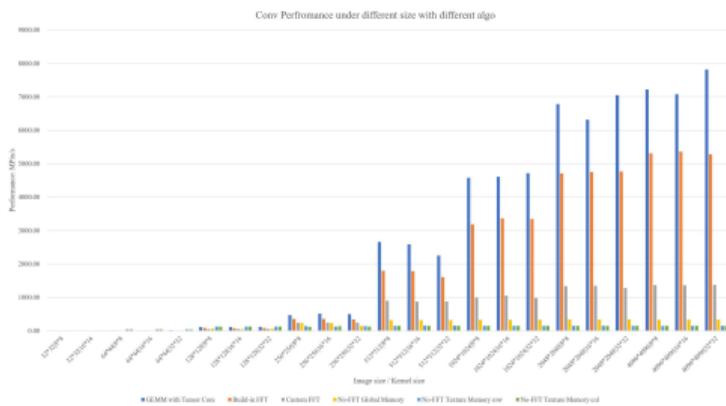


图: 不同方法的卷积计算性能 (大尺寸图像/小尺寸图像)

大尺寸图像下基于 GEMM 的方法优势明显，小尺寸图像下基于纹理内存的方法优势明显。



Benchmark:: 卷积

使用 nvprof 和 NSight 进行分析：可见对于 GEMM，主要瓶颈在于访存。而对于纹理内存的直接方式，主要瓶颈在于计算本身。

表：快速傅里叶变换/纹理内存直接方式计算卷积的对比

类型	计算方法	项目	小尺寸用时占比	大尺寸用时占比
GPU 活动	FFT	内存复制	8.78%	62.91%
		计算	16.88%	4.25%
	直接 (纹理)	内存复制	0.46%	1.88%
		计算	99.19%	98.12%
API 调用	FFT	内存分配、释放	99.09%	96.09%
		设备同步	0.12%	1.30%
	直接 (纹理)	内存分配、释放	77.65%	33.69%
		设备同步	1.00%	55.01%
		上下文切换	19.51%	9.39%



CUDA C++ 卷积神经网络

完整的卷积神经网络涉及三部分，如表所示。本文将考察开启和关闭张量核心下三部分的性能变化。

表: 卷积神经网络中三部分计算任务及特征

过程	cuDNN::cuDNNConvolution-	描述
前向传播	Forward()	包含大量卷积、混合矩阵运算
反向传播更新卷积核权重	BackwardFilter()	包含大量梯度、浮点数值计算，涉及卷积
反向传播更新连接权重	BackwardData()	包含大量梯度、浮点数值计算



CUDA C++ 卷积神经网络

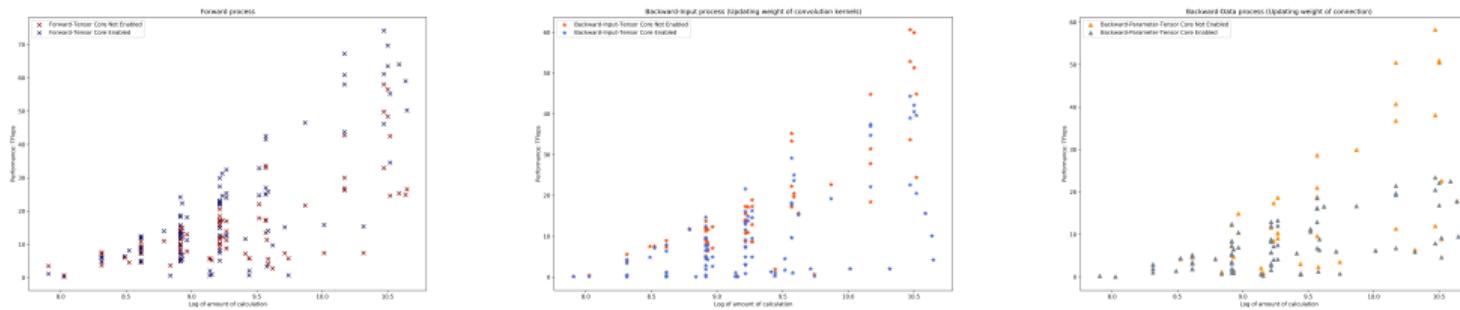
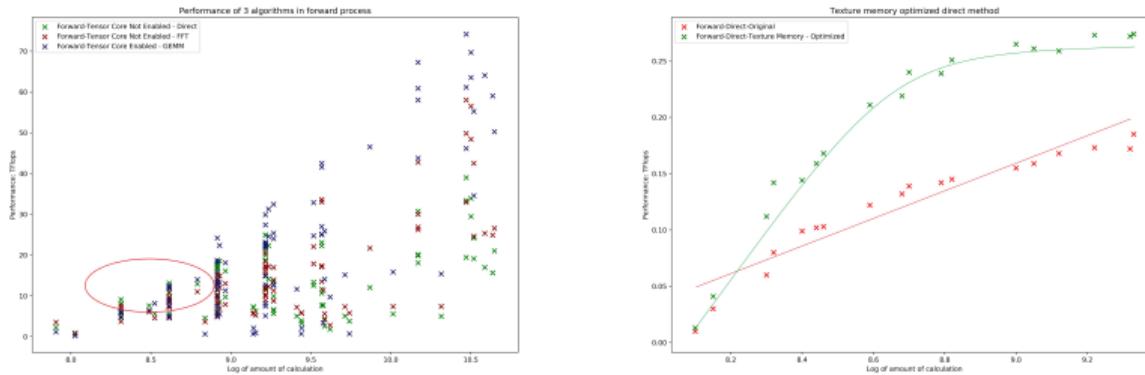


图: 不卷积神经网络不同过程的性能

除存在密集矩阵乘加、卷积运算的前向传播，其余部分开启张量核心后性能均有下降。由上节图像尺寸较小时使用基于纹理内存的直接计算方法取得较好的结果，尝试使用直接计算方法以及纹理内存进行优化。



CUDA C++ 卷积神经网络



图：使用直接方法辅以纹理内存优化的卷积

可见在图像尺寸较小时该优化方式效果较好，随着图像尺寸增大，纹理内存的性能逐渐到达瓶颈。

* 不使用纹理内存直接进行计算的理由为 cuDNN 封装底层计算，故采用这种间接比较的方式。



CUDA C++ 支持向量机

最小序列优化支持向量机中涉及部分矩阵运算，且由于支持向量机特征向量的特点，使用稀疏矩阵进行优化会有较好的效果，本文采用了如下方法进行支持向量机中的矩阵运算。

表: SMO-SVM 中的矩阵计算方式

矩阵计算方法	描述
GEMM Legacy	基于老架构、老 SDK 的 GEMM 实现方式
GEMM	基于新架构、新 SDK 的不使用张量核心的实现方式
GEMM Tensor Core	基于新架构、新 SDK 的使用张量核心的实现方式
cuSPARSE	基于稀疏矩阵库的实现方式



CUDA C++ 支持向量机

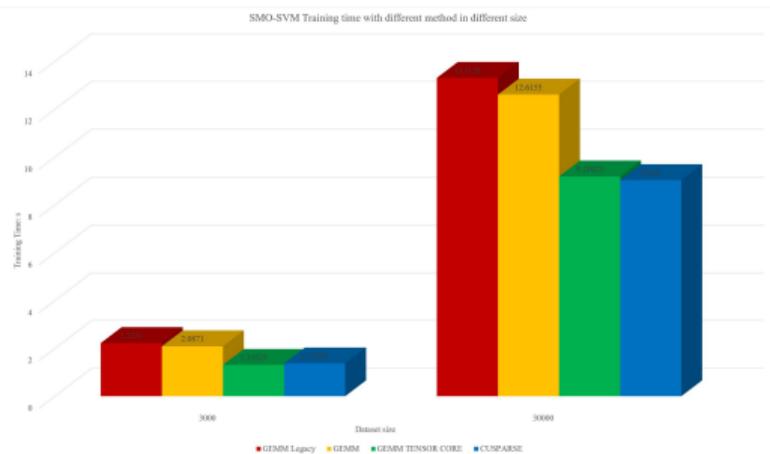


图: SMO-SVM 训练时间

在数据条目较多、维度较高时由于特征较为稀疏，CSR 压缩方式能取得较大压缩比，使用 CUDA 中 cuSPARSE 中为稀疏矩阵优化的 API 能取得较高的性能，而条目较小时 CSR 压缩方式压缩比较小，此时使用张量核心进行 GEMM 运算能取得较好的效果。**那么能否进一步对 GEMM 指令 wmma 进行稀疏矩阵优化呢**



TensorRT 与 Jetson 优化推理

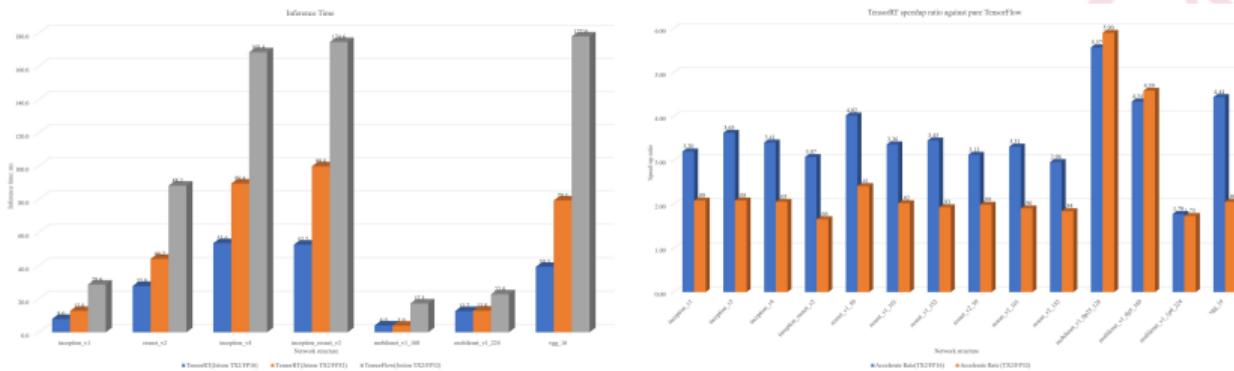
之前的章节阐述了机器学习应用训练阶段新架构 GPU 的性能提升的研究与评估。而神经网络广泛用于嵌入式设备，对推理延迟要求较高，故本节考察不同网络结构采用 TensorRT 进行优化，采用 Jetson TX2 作为目标硬件进行评估。网络结构及特征如下表所示。

表：网络推理实验中使用的网络及特点

网络结构	描述
Inception_v1	网络深度、宽度较大，总体计算量较大
ResNet_v2	引入残差解决梯度、过拟合问题，网络可以更深
Inception_v3	引入批归一化，使用 $1 \times n, n \times 1$ 代替 $n \times n$ 卷积，卷积形状不同
Inception_v4	使用残差优化的 Inception_v3
Inception_resnet_v2	结合 Inception 与 ResNet 而生，与 Inception_v4 较相似
mobileNet_v1	压缩网络规模，将标准卷积分解为深度卷积和逐点卷积
vgg_16	结构简化，但参数数量异常巨大，达 1.38 亿



TensorRT 与 Jetson 优化推理



图：使用 TensorRT 优化后的推理延迟及相应加速比

可见结构简单、参数量巨大的 vgg_16 采用 TensorRT 优化后效果较好。而将卷积拆分为逐点和深度的 mobileNet 由于卷积形状与张量核心不匹配，在半精度下提升比单精度低。然而由于网络本身已经极大压缩，推理速度已经极快。



Tensor Flow-GPU::LeNet-5 卷积神经网络

最后，本文采用最贴近真实应用场景的，基于 Tensor Flow 框架搭建的 LeNet-5，结合上文实验结果进行优化。其中黄色与红色会带来较明显的精度下降，蓝色为原始参数。

表: 基于 Tensor Flow 框架的 CNN 的优化结果

基准时间	超参数	更改方式	倍率	精度	倍率	卷积方式	倍率
54.016(s)	批大小	16	$\times 17.07$	FP32	$\times 1$	原始 (GEMM)	$\times 1$
		32	$\times 5.21$	FP16	$\times 0.86$	纹理	$\times 0.94$
		64	$\times 2.10$	INT8	$\times 0.82$	FFT	$\times 1.11$
		128	$\times 1$				
		256	$\times 0.51$				
		512	$\times 0.31$				
		1024	$\times 0.21$				
		5×5	$\times 1$				
		8×8	$\times 0.97$				



总结

- 张量核心在操作数形状、尺寸与硬件参数、调用特征较为匹配的情况下能通过以精度换速度的策略取得较高的性能提升。
- 张量核心在图像尺寸较大时进行卷积运算优势明显，而图像尺寸较小时使用纹理内存的直接方法优势明显。
- 由于卷积神经网络连接、每层卷积核参数变化较多，故张量核心提升有限。
- 使用 GPU 进行计算时应根据任务特征，如 SMO-SVM 中的稀疏矩阵优化。
- 基于 Tensor Flow-GPU 框架的应用可以通过调整精度、更改超参数、重写卷积计算方式提升性能。
- TensorRT 能极大优化训练完毕的网络，为推理带来极大提升。

截至目前为止，新架构硬件仍然对问题规模、计算方法、数据分布等特征较敏感，实际使用时应根据具体情况权衡。



展望

- 更大规模的矩阵乘加指令
- 使用稀疏矩阵优化的矩阵乘加指令
- 更细粒度的同步机制