# 利用TensorRT加速gpu上的TensorFlow推理

发布者：

Siddharth Sharma - NVidia Sami Kama技术产品营销经理 - NVidia Julie Bernauer深度学习开发人员技术专家 - 追求工程解决方案架构师，NVidia Laurence Moroney开发人员倡导者，谷歌

### 概述

TensorFlow仍然是当今最流行的深度学习框架，在全球拥有数万用户。NVIDIA®TensorRT™是一个深度学习平台，它优化了神经网络模型，并加快了在数据中心、嵌入式和汽车设备中运行的GPU加速平台的推理速度。我们对TensorFlow和TensorRT的集成感到兴奋，这似乎是一个自然的契合，特别是NVIDIA提供了非常适合加速TensorFlow的平台。这使得TensorFlow用户在使用TensorRT时具有极高的推理性能和近乎透明的工作流。

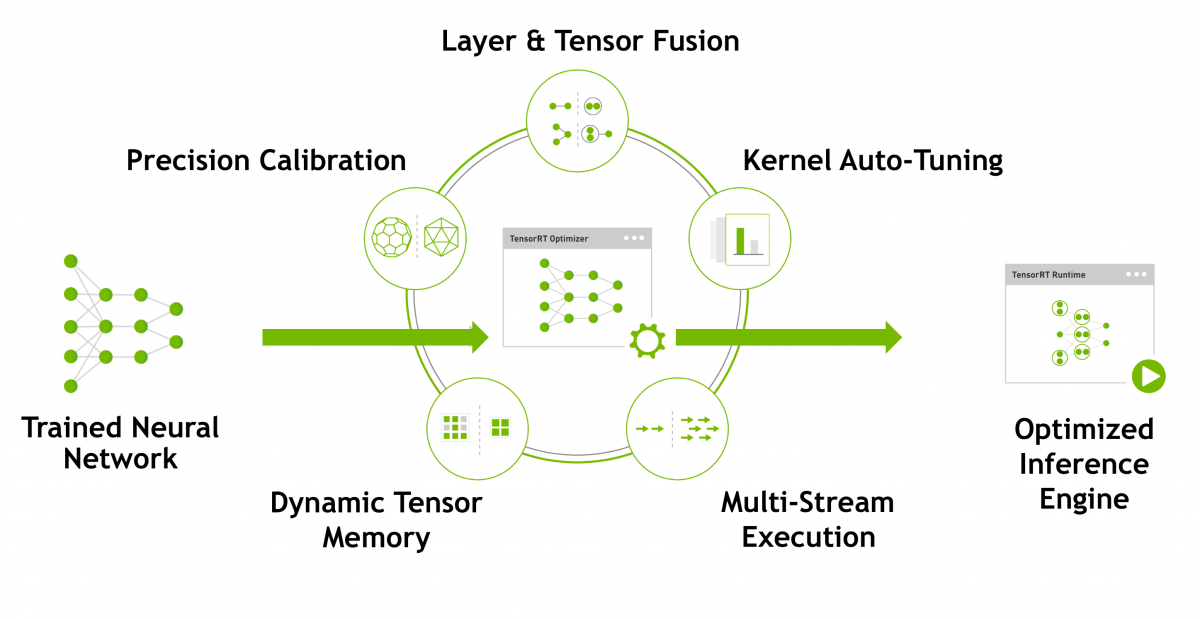


图1。TensorRT优化经过训练的神经网络模型，以生成部署就绪的运行时推理引擎。

TensorRT对神经网络图进行了几个重要的转换和优化（图2）。首先，消除未使用输出的层，以避免不必要的计算。接下来，在可能的情况下，卷积层、偏压层和ReLU层被融合形成一个单层。另一个转换是水平层融合或层聚合，以及聚合层到其各自输出所需的划分。水平层融合通过合并具有相同源张量和应用具有相似参数的相同操作的层来提高性能。请注意，这些图优化不会更改图中的底层计算：相反，它们希望重新构造图以更快、更高效地执行操作。

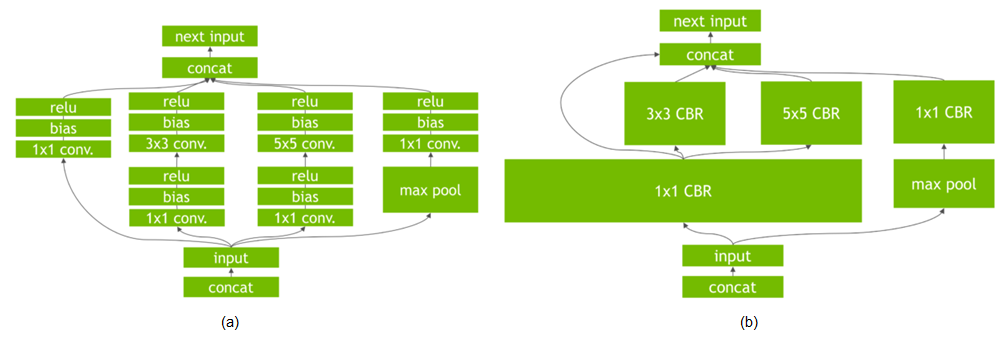


图2（a）：具有多个卷积层和激活层的卷积神经网络示例。（b）TensorRT的垂直和水平层融合和层消除优化简化了GoogLeNet初始模块图，减少了计算和内存开销。

如果您已经在TensorFlow模型中使用TensorRT，那么您知道应用TensorRT优化需要导出经过训练的TensorFlow图。您还需要手动导入某些不受支持的TensorFlow层，然后在TensorRT中运行完整的图。在大多数情况下，你不应该再这样做了。在新的工作流中，使用一个简单的API从TensorFlow中使用TensorRT应用强大的FP16和INT8优化。现有的反编译程序只需要几个新的代码行来应用这些优化。

对于ResNet-50基准的低延迟运行，TensorRT将TensorFlow推断速度提高了8倍。这些性能改进只需花费几行额外的代码，就可以使用TensorFlow1.7及更高版本。在本文中，我们将描述新的工作流和api，以帮助您开始使用它。

### TensorRT优化在TensorFlow图中的应用

向TensorFlow推理工作流添加TensorRT需要一个额外的步骤，如图3所示。在这一步中（以绿色突出显示），TensorRT从冻结的TensorFlow图构建优化的推理图。

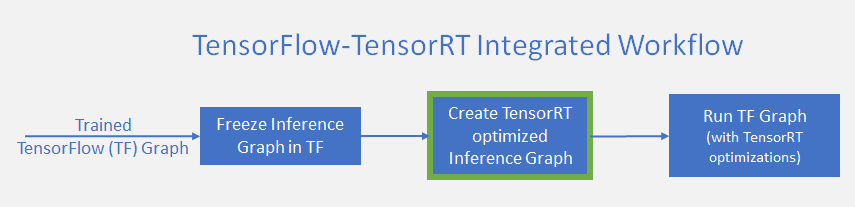


图3：在TensorFlow中使用TensorRT时的工作流图

为了实现这一点，TensorRT获取冻结的TensorFlow图并对其进行解析，以选择它可以优化的子图。然后，它对子图应用优化，并将其替换为原始TensorFlow图中的TensorRT节点，使其余图保持不变。在推理过程中，TensorFlow执行调用TensorRT的完整图来运行TensorRT优化的节点。通过这种方法，开发人员可以继续使用TunSoRT的优化来使用灵活的TysFooSoad特征集。

让我们看一个有三个段a、B和C的图的例子。TensorRT优化了段B，然后用一个节点替换它。在推断过程中，TensorFlow执行A，调用TensorRT执行B，然后TensorFlow执行C。从用户的角度来看，您将继续使用TensorFlow。

TensorRT优化TensorFlow图中可能的最大子图。子图中的计算越多，从TensorRT得到的好处就越大。为了获得最佳性能，您需要优化大多数图，并用最少的TensorRT节点替换它们。基于图中的操作，最终的图可能有多个TensorRT节点。使用TensorFlow API，可以指定子图中要转换为TensorRT节点的最小节点数。任何节点数小于指定集合的子图，即使与TensorRT兼容，也不会转换为TensorRT引擎。这对于包含由不兼容节点分隔的小型兼容子图的模型非常有用，进而导致小型TensorRT引擎。

让我们更详细地看看如何实现工作流。

### 使用新的TensorFlow api

新的TensorFlow API支持用几行新代码直接实现TensorRT优化。首先，指定TensorFlow允许使用的可用GPU内存的一部分，其余内存可用于TensorRT引擎。这可以通过GPUOptions函数的新per\_process\_gpu\_memory\_fraction参数来完成。此参数需要在TensorFlow TensorRT进程第一次启动时设置。例如，将per\_process\_gpu\_memory\_fraction设置为0.67将为TensorFlow分配67%的gpu内存，为TensorRT引擎分配剩余的第三个gpu内存。

下一步是让TensorRT分析TensorFlow图，应用优化，并用TensorRT节点替换子图。您可以使用新的create\_inference\_graph函数对冻结图应用TensorRT优化。此函数使用冻结的TensorFlow图作为输入，然后返回带有TensorRT节点的优化图，如以下代码段所示：

让我们看看函数的参数：

输入图：冻结张量流图

输出：带有输出节点名称的字符串列表，例如。[“resnet\_v1\_50/predictions/respect\_1”]

最大批大小：整数，输入批的大小，例如16

Max工作空间：整数，最大的GPU内存大小可用于TySRRT

精度模式：字符串，允许值“FP32”、“FP16”或“INT8”

最小分段大小：整数（默认值=3），控制要创建的TensorRT引擎的子图中的最小节点数

PuryPrimeGPUMyMyLyMyStand和Max WorkStudioSysiZyByb参数应该一起使用，以分离TensorFlow和TensorRT之间的GPU内存，以提供最佳的整体应用性能。为了最大化推理性能，您可能希望给TunSoRT稍微多一些内存，而不是它所需要的内存。剩下的给TensorFlow。例如，如果将per\_process\_gpu\_memory\_fraction参数设置为（12–4）/12=0.67，则将12GB gpu的max\_workspace\_size\_bytes参数设置为4000000000将为TensorRT引擎分配~4GB。同样，找到最佳内存分割依赖于应用程序，可能需要一些迭代。

### Using TensorBoard to Visualize Optimized Graphs

TensorBoard enables us to visualize the changes to the ResNet-50 node graph once TensorRT optimizations are applied in TensorBoard. Figure 4 shows that TensorRT optimizes almost the complete graph, replacing it with a single node titled “**my\_trt\_op0**” (highlighted in red). Depending on the layers and operations in your model, TensorRT nodes replace portions of your model due to optimizations. The box titled “conv1” isn’t actually a convolution layer; it’s really a transpose operation from NHWC to NCHW.

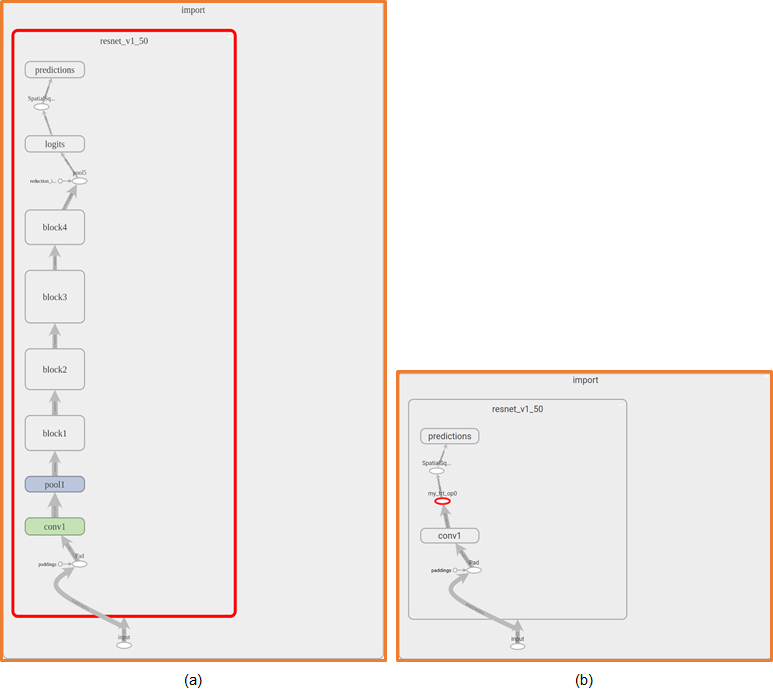
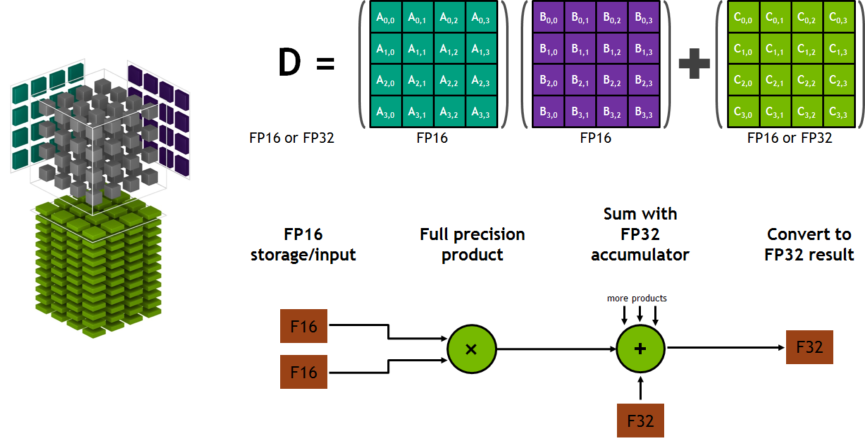


Figure 4. (a) ResNet-50 graph in TensorBoard (b) ResNet-50 after TensorRT optimizations have been applied and the sub-graph replaced with a TensorRT node.

### 在Volta GPUs上使用张量核

与FP32或FP64相比，使用半精度（也称为FP16）算法减少了神经网络的内存使用。FP16支持部署更大的网络，同时占用的时间少于FP32或FP64。NVIDIA的Volta架构结合了硬件矩阵数学加速器Tensor Cores。张量核提供一个4x4x4矩阵处理阵列，它执行D=a\*B+C操作，其中a、B、C和D是4×4矩阵。图5显示了这是如何工作的。矩阵乘法输入A和B是FP16矩阵，而累积矩阵C和D可以是FP16或FP32矩阵。



无花果

图5：张量核上的矩阵处理操作

TensorRT在使用FP16数学时，当检测到用于推理的硬件Tensor核时，会自动使用硬件Tensor核。Tensor Cores在NVIDIA Tesla V100上的峰值性能比双精度（FP64）快一个数量级，而吞吐量比单精度（FP32）快4倍。只需使用“FP16”作为create\_inference\_graph函数中precision\_mode参数的值，即可启用半精度，如下所示。getNetwork（）是一个帮助函数，它从protobuf文件读取冻结的网络并返回网络的atf.GraphDef（）。

图6显示ResNet-50使用NVIDIA Volta Tensor内核与TensorFlow TensorRT集成相比，在7 ms延迟下执行速度快8倍，而只在相同硬件上运行TensorFlow。

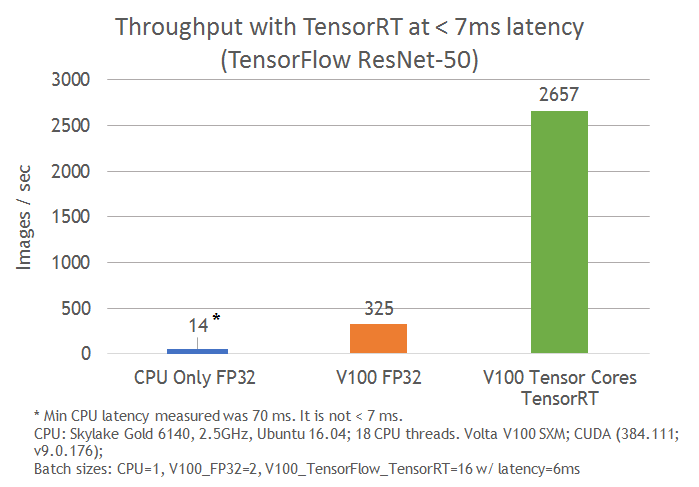
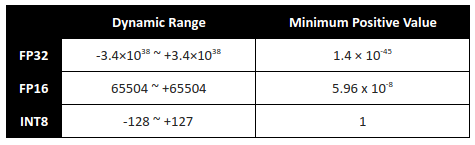


图6:ResNet-50推理吞吐量性能

### 使用INT8精度进行推理

使用INT8精度进行推理进一步提高了计算速度，对带宽的要求也较低。动态范围的缩小使得神经网络的权值和激活具有挑战性。



TensorRT提供的功能可以接受单精度（FP32）和半精度（FP16）训练的模型，并将其转换为使用INT8量化进行部署，同时将精度损失降至最低。在应用前面描述的TensorRT优化之前，使用INT8转换部署模型需要校准经过训练的FP32模型。工作流更改为在创建TensorRT优化推理图之前包含校准步骤，如图7所示：

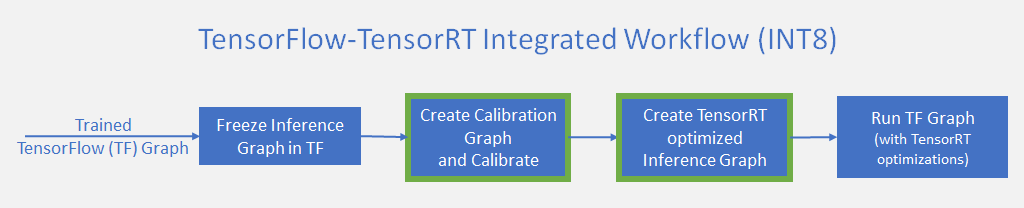


图7。包含INT8推理的工作流

首先使用create\_inference\_graph函数，将precision\_mode参数设置为“INT8”来校准模型。此函数的输出是一个冻结的TensorFlow图，可以进行校准。

现在用校准数据运行校准图。TensorRT使用节点数据的分布来量化节点的权重。必须使用校准数据来密切反映生产中问题数据集的分布。我们建议在第一次使用用INT8校准的模型时，检查推理过程中的误差累积。最小分段大小参数有助于优化图以最小化量化误差。使用minimum\_segment\_size，您可以更改优化的INT8引擎中的最小节点数，以更改最终优化的图形，从而微调结果精度。

在对校准数据执行图形后，使用calib\_graph\_to\_infer\_graph函数对校准图形应用TensorRT优化。此函数还用为INT8优化的TensorRT节点替换TensorFlow子图。函数的输出是一个冻结的张量流图，可以像往常一样用于推理。

只需要这两个命令就可以用TensorFlow模型实现INT8精度推断。

如果要查看此处显示的示例，请查看运行这些示例所需的代码

### 可利用性

我们预期，当使用Nvidia GPU时，TySoRT与TensorFlow的集成将获得最高性能，同时保持TySoFrof的易用性和灵活性。NVIDIA继续与Google TensorFlow团队密切合作，进一步增强这些集成功能。开发人员将自动受益于更新，因为TunSRRT支持更多的网络，而不需要更改现有代码。

Find instructions on how to get started today at: <https://www.tensorflow.org/install/install_linux>

In the near future, we expect the standard pip install process to work as well. Stay tuned!

We believe you’ll see substantial benefits to integrating TensorRT with TensorFlow when using GPUs. You can find more information on TensorFlow at <https://www.tensorflow.org/>.

Additional information on TensorRT can be found on NVIDIA’s TensorRT page at <https://developer.nvidia.com/tensorrt>.