文章编号：1002-2082 (2020) 02-0337-05

使用 TensorRT 进行深度学习推理

周立君[1]，刘 宇[1]，白 璐[2]，刘 飞[1]，王亚伟[1]

（1. 西安应用光学研究所，陕西 西安 710065；2. 西安北方光电科技防务有限公司，陕西 西安 710043）

摘 要： TensorRT 是一个高性能的深度学习推理平台。它包括一个深度学习推理优化器和运行时为深度学习推理应用程序提供低延迟和高吞吐量。给出了一个使用 TensorRT 快速构建计算管道的例子，实现通过 TensorRT 执行智能视频分析的典型应用。该示例演示了使用片上解码器进行解码、使用片上标量进行视频缩放和 GPU 计算的 4 个并发视频流。为了演示的简单性，只有一个通道使用 NVIDIA TensorRT 执行对象标识，并在标识的对象周围生成包围框。该示例还使用视频转换器函数进行各种格式转换，使用 EGLImage 来演示缓冲区共享和图像显示。最后采用 GPU 卡 V100 对 ResNet 网络进行 TensorRT 加速性能的实际测试，结果表明 TensorRT 能够使吞吐量提升大约 15 倍。

关键词： TensorRT；深度学习推理；对象检测；统一计算设备架构

中图分类号： TN219　　**文献标志码：** A　　**DOI：** 10.5768/JAO202041.0202007

Using TensorRT for deep learning and inference applications

ZHOU Lijun[1], LIU Yu[1], BAI Lu[2], LIU Fei[1], WANG Yawei[1]

（1. Xi’an Institute of Applied Optics, Xi’an 710065, China; 2. Xi’an North Electro-optic Science & Technology CO.LTD., Xi’an 710043, China）

Abstract: TensorRT is a high-performance deep learning and inference platform. It includes a deep learning and inference optimizer as well as runtime that provides low latency and high throughput for deep learning and inference applications. An example of using TensorRT to quickly build computational pipelines to implement a typical application for performing intelligent video analysis with TensorRT was presented. This example demonstrated four concurrent video streams that used an on-chip decoder for decoding, on-chip scalar for video scaling, and GPU computing. For simplicity of presentation, only one channel used NVIDIA TensorRT to perform object identification and generate bounding boxes around the identified objects. This example also used video converter functions for various format conversions, EGLImage to demonstrate buffer sharing and image display. Finally, the GPU card V100 was used to test the TensorRT acceleration performance of ResNet network. The results show that TensorRT can improve the throughput by about 15 times.

Key words: TensorRT; deep learning and inference; object identification; compute unified device architecture

引言

英伟达 TensorRT 是一种高性能神经网络推理（Inference）引擎，是一个标准 C++库。TensorRT 只能用来做 Inference（推理），不能用来进行训练，用于在生产环境中部署深度学习应用程序[1]。应用领域包括图像分类、分割和目标检测等，可提供最大的推理吞吐量和效率。TensorRT 需要 CUDA（compute unified device architecture）的支持，包含一个为优化生产环境中部署的深度学习模型而创建的库，可获取经过训练的神经网络（通常使用 32 位或 16 位数据），并针对降低精度的 INT8 运算来优化这些网络。借助 CUDA 的可编程性，TensorRT 将能够应对

收稿日期： 2019-06-17；　**修回日期：** 2019-09-20

基金项目： 装备预先研究兵器工业联合基金（6141B01020205）

作者简介： 周立君（1982–），男，硕士，高级工程师，主要从事人工智能技术研究。E-mail: 376187676@qq.com

深度神经网络日益多样化、复杂化的趋势。TensorRT 在不断的改进过程中，在保证软件精度的同时，自动优化训练过的神经网络，不断提高速度。TensorRT 能够支持 Caffe 等主流深度学习框架[2-3]。

本文实现了一个利用 TensorRT 执行智能视频分析的典型应用，演示了使用片上解码器进行解码，使用片上转换器进行视频缩放，利用 TensorRT 执行对象标识，利用 OpenGL2 和 XWindow-11 进行渲染，并在标识的对象周围生成包围框。此外，还使用视频转换器函数进行各种格式转换。使用 EGLImage 来演示缓冲区共享和图像显示。图 1 展示了使用 TensorRT 的流程细节[4-5]。

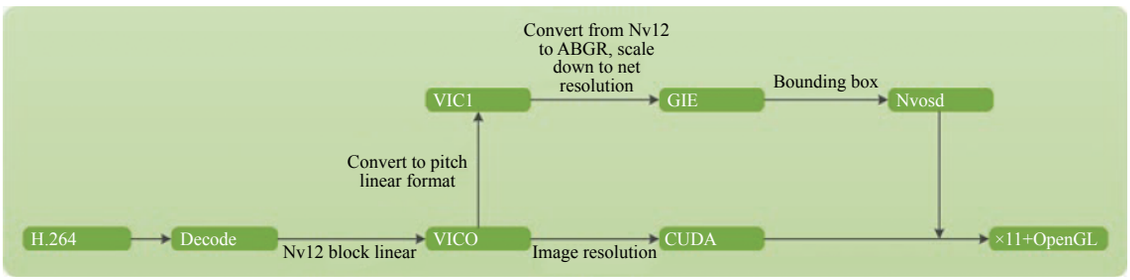

图 1 TensorRT 处理流程

Fig. 1 TensorRT processing flow

1 定义视频处理结构体

本文例程将本地存储的 H.264 视频文件进行解码、格式转换和渲染，为了使流程清晰，便于管控，定义 context_t 结构管理视频处理全部资源。如表 1 所示成员主要包括一个解码器类、一个转换器类、一个渲染器类、一个数据指针。解码器类 NvVideoDecoder 封装了用于视频解码的多媒体 API 函数，用于从 H.264 视频文件解码压缩的视频。转换器类 NvVideoConverter 封装了视频转换的相关函数，包括色彩空间变换、尺度变换以及软硬件缓存空间的变换。渲染器类 NvEglRenderer 类封装了图像渲染的相关函数以及 XWindow-11 以及 OpenGL2 的部分函数，使用 EGL 和 OpenGL ES 2.0 进行呈现。渲染器需要缓冲区的文件描述符(FD)作为输入，创建自己的 X 窗口。渲染速率(以帧/s 为单位)，窗口的宽度、高度、水平偏移量和垂直偏移量都是可配置的。所有 EGL 调用只能通过一个线程进行。该类在内部创建一个线程，该线程执行所有 EGL/GL 初始化，从 FD 获取 EGLImage 对象，然后反初始化所有 EGL/GL 结构[6-9]。

表 1 Context_t 结构主要成员

Table 1 Main members of Context_t structure

成员	描述
NvVideoDecoder	包含视频解码相关的成员和函数
NvVideoConverter	包含视频格式转换相关的成员和函数
NvEglRenderer	包含EGL显示渲染相关函数
EGLImageKHR	EGLImage图像数据指针，用于CUDA处理，这个类型来源于EGL开源库

2 利用 TRT_Context 类进行加速推理

英伟达提供的 TRT_Context 类包含一系列接口来加载 Caffe 模型并执行推理。表 2 描述了本示例中使用的关键 TRT_Context 成员。本文中使用 buildTrtContext 实现 Caffe 模型到 TensorRT 模型转换，它的输入参数包括 caffe 网络结构文件和模型参数文件。实际上也可借助转换工具实现 Caffe 模型到 gie 模型的转换。TRT_Context::getNumTrtInstances 用于获取加速上下文的实例。TRT_Context::doInference 用转换好的模型利用 TensorRT 进行加速推理。此外，TRT_Context 还实现了一些模型控制和剪裁的一些函数接口[10-13]。

表 2 TRT_Context 类主要成员

Table 2 Main members of TRT_Context class

TRT_Context类成员	描述
TRT_Context::buildTrtContext	构建Tensorrt上下文
TRT_Context::getNumTrtInstances	获取TRT_context 实例.
TRT_Context::doInference	TensorRT 推理接口

3 主进程

主进程调用以上定义的类和结构实现整个处理流程，主要代码如下：

```
TRT_Context g_trt_context;
main(int argc, char *argv[])
{
  //程序入口参数处理
  context_t ctx[CHANNEL_NUM];
```

```
global_cfg cfg;
char **argp;
set_globalcfg_default(&cfg);
argp = argv;
parse_global(&cfg, argc, &argp);
parse_csv_args(&ctx[0], &g_trt_context, argc-cfg.channel_num-1, argp);
//设置 g_trt_context 参数
g_trt_context.setModelIndex(TRT_MODEL);
g_trt_context.buildTrtContext(cfg.deployfile,cfg.modelfile, true);
pthread_create(&TRT_Thread_handle,NULL,trt_thread, NULL);
// 获取 EGL 默认值
egl_display = eglGetDisplay(EGL_DEFAULT_DISPLAY);
// EGL 初始化
eglInitialize(egl_display, NULL, NULL)
for (iterator = 0; iterator < cfg.channel_num; iterator++)
{
    int ret = 0;
    sem_init(&(ctx[iterator].dec_run_sem), 0, 0);
    set_defaults(&ctx[iterator]);
    char decname[512];
    sprintf(decname, "dec%d", iterator);
    ctx[iterator].channel = iterator;
    ctx[iterator].in_file_path = cfg.in_file_path[iterator];
    ctx[iterator].nvosd_context = nvosd_create_context();
    //创建解码器
    ctx[iterator].dec = NvVideoDecoder::createVideoDecoder(decname);
        //设置输出面板格式
        ctx[iterator].dec->setOutputPlaneFormat(ctx[iterator].decoder_pixfmt, CHUNK_SIZE);
    //映射输出面板缓存
        ctx[iterator].dec->output_plane.setupPlane(V4L2_MEMORY_MMAP, 10, true, false);
    //创建渲染线程
        pthread_create(&ctx[iterator].render_feed_handle, NULL, render_thread, &ctx[iterator]);
    char convname[512];
    // 创建 BL 到 PL 转换器
    ctx[iterator].conv = NvVideoConverter::createVideoConverter(convname);
ctx[iterator].conv->output_plane.setDQThreadCallback(conv_output_dqbuf_thread_callback);
ctx[iterator].conv->capture_plane.setDQThreadCallback(conv_capture_dqbuf_thread_callback);
    if (ctx[iterator].cpu_occupation_option!=PARSER)
    pthread_create(&ctx[iterator].dec_capture_loop, NULL, dec_capture_loop_fcn, &ctx[iterator]);
    pthread_create(&ctx[iterator].dec_feed_handle, NULL, dec_feed_loop_fcn, &ctx[iterator]);
    //等待解码器获取 EOS
    sem_wait(&(ctx[iterator].dec_run_sem));
    //向渲染器发送命令
    ctx[iterator].stop_render = 1;
    pthread_cond_broadcast(&ctx[iterator].render_cond);
    pthread_join(ctx[iterator].render_feed_handle, NULL);
    }
}
```

4　测试与分析

在这个示例中, 对象检测仅限于在 960×540 分辨率的视频流中识别汽车。该网络基于 GoogleNet。推理是在逐帧的基础上进行的, 不涉及任何对象跟踪, 展示了如何使用 TensorRT 快速构建计算管道。示例使用训练过的 GoogleNet 网络, 它是用 NVIDIA 深度学习 GPU 训练系统(DIGITS)训练的。训练是大约 3 000 帧从 1.5 m(5 英尺)~3 m(10 英尺)的高度拍摄的。根据输入的视频样本, 预计会有不同程度的检测精度。运行程序对 H.264 本地视频进行测试, TensorRT 能够成功运行, 实时识别目标图像[14-15], 测试效果如图 2 所示。

其运行性能如下:

```
FP32 run:400 batches of size 100 starting at 100
.....
Top1: 0.9904, Top5: 1
Processing 40000 images averaged 0.00157702 ms/image and 0.157702 ms/batch.
FP16 run:400 batches of size 100 starting at 100
```

```
Engine could not be created at this precision
INT8 run:400 batches of size 100 starting at 100
.....................................
Top1: 0.9908, Top5: 1
Processing 40000 images averaged 0.00122583 ms/image and 0.122583 ms/batch.
```

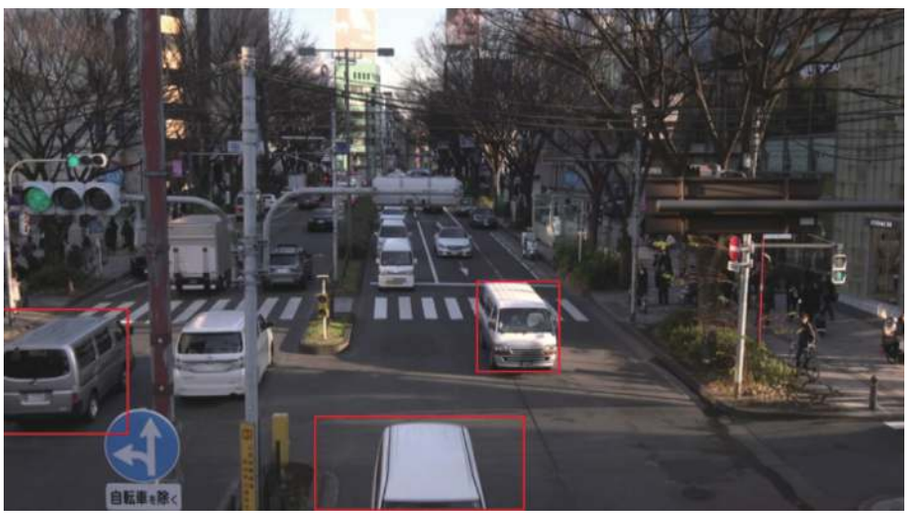

图 2　程序测试效果

Fig. 2　Program test effect

可以看到这个例子中采用 int8 量化时，提速可以达到 20% 以上，对于大计算量的应用，提速效果更好。推理（Inference）可以使用低精度的技术，训练的时候因为要保证前后向传播，每次梯度的更新是很微小的，这个时候需要相对较高的精度，一般来说需要 float 型，如 FP32，32 位的浮点型来处理数据。但是在推理的时候，对精度的要求没有那么高，很多研究表明可以用低精度，如半长（16 字节）的 FP16，也可以用 8 位的整型 INT8 来做推理，结果没有特别大的精度损失。低精度计算的好处是一方面可以减少计算量，原来计算 32 位的单元处理 FP16 的时候，理论上可以达到 2 倍的速度，处理 INT8 的时候理论上可以达到 4 倍的速度。另一方面是模型需要的空间减少，不管是权值的存储还是中间值的存储，应用更低的精度，模型大小会相应减小。

TensorRT 的运行效果与 GPU 的硬件性能和采用的网络结构直接相关，量化标准仅仅是其中一个影响因素，不同的硬件和网络结构也会带来不同程度的速度提升。为了对比上述例子的加速性能，图 3 展示了采用 GPU 卡 V100 对 ResNet 网络进行 TensorRT 加速的实际效果。

这是一个比较极端的例子，该例中使用的是先进的 GPU 卡 V100，V100 添加了专门针对深度学习优化的 TensorCore，TensorCore 可以完成 4×4 矩

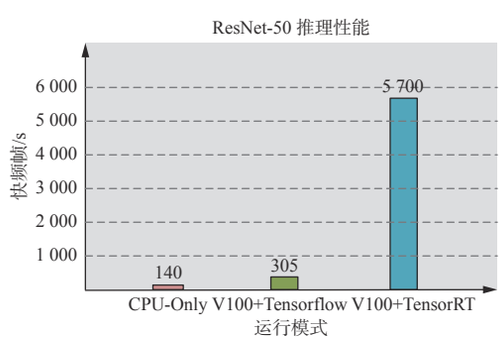

图 3　V100 卡+ResNet 网络使用 TensorRT 推理的效果

Fig. 3　Effect of TensorRT reasoning in V100 card+ResNet network

阵的半精度乘法，也就是可以完成一个 4×4 的 FP16 矩阵和另外一个 4×4 的 FP16 矩阵相乘，当然可以再加一个矩阵（FP16 或 FP32），得到一个 FP32 或者 FP16 的矩阵的过程。TensorCore 在 V100 上理论峰值可以达到 120 Tflops。如果只是用 CPU 来做推理，首先它的吞吐只能达到 140，也就是说每秒只能处理 140 张图片，同时整个处理过程需要有 14 ms 的延迟，也就是说用户提交请求后，推理阶段最快需要 14 ms 才能返回结果；如果使用 V100，在 TensorFlow 中去做推理，大概是 6.67 ms 的延时，但是吞吐只能达到 305；如果使用 V100 加 TensorRT，在保证延迟不变的情况下，吞吐可以提高 15 倍，高达 5 700 张图片帧/s。可以看到随着 GPU 性能的提升，以及网络结构的复杂化，TensorRT

对推理速度的提升非常明显, 对于大数据应用是一个很好的选择。目标英伟达公司已经将 TensorRT 项目部分开源, 这势必会使 TensorRT 得到更好的推广应用。

参考文献:

[1] NVIDIA. NVIDIA Deep learning SDK[DB/OL]. [2019-11-27] https://docs.nvidia.com/deeplearning/sdk/index.html.

[2] HINTON G E. Where do features come from?[J]. Cognitive Science, 2014, 38(6): 1078-1101.

[3] BISHOP C.Neutal networks for pattern recognition[M]. London: Oxford University Press, 1995.

[4] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86:2278-2324.

[5] RICHARD S, CHRISTOPHER M, ANDREW N. Learning continuous phrase representations and syntactic parsing with recursive neural networks[C]//Taboe Nevada: MPS Neural Information Processing System, 2010: 1-9.

[6] CHEN Y, CHEN T, X UZ, et al. Diannao family: energy-efficient hardware accelerators for machine learning[J]. Communications of the ACM, 2016, 59(11): 105-112.

[7] SUN F, WANG C, GONG L, et al. A power-efficient accelerator for convolutional neural networks[C]//2017 IEEE International Conference on Cluster Computering (CLUSTER). Washington: IEEE, 2017: 631-632.

[8] LUO T, LIU S, LI L, et al. Dadiannao: a neural network supercomputer[J]. IEEE Transactions on Computers, 2017, 66(1): 73-88.

[9] GOKEN E, ERASLAN Z A, JULIEN G, et al. Deep learning: new computational modelling techniques for genomics[J]. Nature Reviews Genetics, 2019, 20(7): 389-403.

[10] AGARWAL A, DUCHI J C. Distributed delayed stochastic optimization[C]//Nevada: MIPS, Neural Information Processing Systems, 2011: 873-881.

[11] YOU Y, ZHANG Z, HSIEH CJ, et al. Imagenet training in minutes [C]//Proceedings of the 47th International Conference on Parallel Processing. New York: NY ACM, 2018: 1-10.

[12] SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676): 354.

[13] LONG J, SHELHAMER E, DARRELL T. Fully convolutional networks for semantic segmentation[C]//Proceeedings of 2015 IEEE Conference on Computer Vision and Pattern Recongnition. Boston, Washington: IEEE, 2015: 3411-3440.

[14] NG Y H, HAUSKNECHT M, VIJAYANARASIMHAN S, et al. Beyond short snippets: deep networks for video classification[J]. IEEE, 2015, 16(4): 4694-4702.

[15] YANG Z, NEVATIA R. A multi-scale cascade fully convolutional network face detector[C]//Pattern Recognition(ICPR), 2016 23rd International Conference on IEEE. Washington: IEEE, 2016: 633-638.