Object Detection with Jetson TX1: Challenges and Insights

Oron Port Gil Shomron
{soronpo, gilsho}@tx.technion.ac.il
Electrical Engineering
Technion — Israel Institute of Technology

ABSTRACT

1. INTRODUCTION

Object detection is a major rule in many different devices, from mobile phones, cameras, and IoT devices, to drones, and autonomous cars. With the aid of machine learning, detecting an object also involves its classification. Fast detection, and accurate classification... I don't know.

GPU is built for throughput. Implemented with a large number of cores (*streaming multiprocessors* (SM)) its parallelism enables algorithms, such as object detection and machines learning, to run faster than on a CPU.

With increasing demand for low-energy modules, NVIDIA started manufacturing an embedded platform called Jetson. We have received the Jetson TX1 to explore the performance of different object detection algorithms.

In this paper we will discuss and analyze the performance of two algorithms: (1) *Single Shot MultiBox Detector* (SSD) [1], as implemented with Caffe, and (2) *You Only Look Once* (YOLO) [2].

2. RESULTS

To compare Caffe SSD performance versus YOLO performance on Jetson TX1, we used their supplied applications for object detection on image files. We downloaded 600 images from MSCOCO dataset [3], and measured the total execution time. We made three comparison between the two algorithms: when both run on (1) CPU, (2) GPU without cuDNN, and (3) GPU with cuDNN[1].

Figure 1 shows the execution time of both SSD and YOLO for each of the options described above. Not surprisingly, running object detection algorithms on a CPU is not as efficient as running them on a GPU. The GPU built-in parallelism is advantageous for such algorithms. [TODO: comparison between SSD and YOLO on CPU]

[TODO: comparison between GPU with and without cuDNN]

[TODO: comparison between SSD and YOLO on GPU]

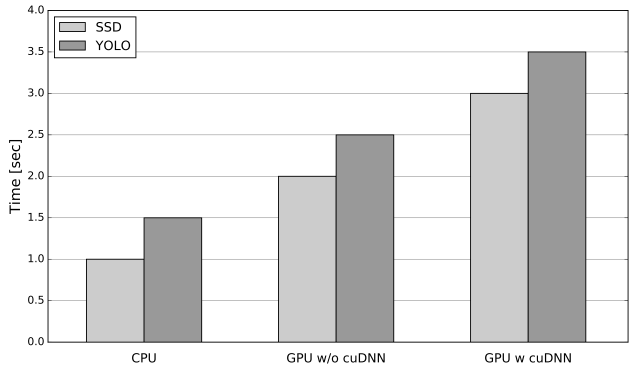


Figure 1: Execution time comparison

3. PROFILING

Figure 2 shows the profiling output of object detection of a single image, using SSD and YOLO. Profiling was done using Nvidia Visual Profiler.

One of Jetson's strengths is that its CPU and GPU share the same memory. Therefore, host-to-device (HtoD) and device-to-host (DtoH) copies can be optimized to zero-copy [4]. Zero-copy will have no effect on SSD and YOLO, since memory copies are not frequent, therefore, there is no speedup potential. In addition, there are no overlapping opportunities between HtoD, DtoH, and the compute kernels, since there are almost no memory copies.

In terms of compute intensity, both algorithms use 75% of their compute time in cuDNN kernels. There is no parallelism between kernels. SSD and YOLO fetch and analyze each image in serial, so theoretically, one can analyze a couple of images in parallel. However, Jetson TX1 has only 2 SMs, with 128 CUDA cores each, therefore, adding additional threads to the system, or running two or more object detection processes in par-

[1]The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

allel, achieves no performance gains.

4. INSTALLATION AND EXECUTION

YOLO compiles out-of-the-box on Jetson TX1. Unfortunately, Caffe does not. In this section we will describe the problems we encountered during installation. Hopefully, this information will ease the process for others. In addition, we will share some tips, gathered during this project, regarding the execution of the SSD implementation.

Makefile.config. To use cuDNN acceleration:

```
USE_CUDNN := 1
```

Tegra X1 has CUDA capability 5.3, therefore append to *CUDA_ARCH*:

```
CUDA_ARCH := -gencode arch=compute_53,
    ↪ code=sm_53
```

HDF5 directories should be added to *INCLUDE_DIRS* and *LIBRARY_DIRS*:

```
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/
    ↪ local/include /usr/include/hdf5/
    ↪ serial
LIBRARY_DIRS := $(PYTHON_LIB) /usr/
    ↪ local/lib /usr/lib/aarch64-linux-
    ↪ gnu/serial
```

Makefile. HDF5 libraries need to be added to the Makefile also:

```
LIBRARIES += glog gflags protobuf
    ↪ boost_system boost_filesystem m
    ↪ hdf5_serial_hl hdf5_serial
```

Python. Caffe also has Python libraries. Running Python scripts can fail due to unset Python path. By running:

```
export PYTHONPATH=$CAFFE_ROOT/python
```

where *$CAFFE_ROOT* is the Caffe home directory, we managed to fix the issues.

Web Camera. Jetson TX1 on-board CSI camera does not work straightaway. On the other hand, plugging a dedicated web-camera almost does. To run Caffe SSD web-camera demo, add the following line before the command:

```
LD_PRELOAD=/usr/lib/aarch64-linux-gnu/
    ↪ libv4l/v4l2convert.so
```

Performance Tuning. The new Tegra Linux driver package releases include *jetson_clocks.sh* script, this is able to maximize performance by disabling DVFS, CPU idle, and CPU quit [5]. To toggle performance:

```
sudo ./jetson_clocks.sh
```

We recommend reading the manual first.

We also noticed that Jetson TX1 fan does not work on default. The script above turns it on. Enabling the fan without running the *jetson_clocks.sh* script, can be achieved with:

```
echo 255 > /sys/kernel/debug/tegra_fan/
    ↪ target_pwm
```

FPS Readings. Both SSD and YOLO produce FPS readings based on the CPU time. Because most computation is done on the GPU, the FPS are readings are incorrect, and need to be measured in another way, e.g., dividing the number of analyzed images by the wall-clock time, or using the profiler.

5. CONCLUSION

SSD and YOLO are not I/O intensive but compute intensive. Since 75% of execution time the algorithms runs cuDNN, which is a well optimized library provided by NVIDIA, we think that in this project time frame it is not possible optimize SSD or YOLO any further. Even if one will optimize the remaining 25%, the 10FPS goal will not be achieved (Amdahl's law).

It is possible to increase the performance of the above algorithms on a Jetson TX1 by modifying the algorithms, probably on the expense of accuracy. For example, Tiny YOLO is a faster version of YOLO. It exhibits 4x speedup over the regular YOLO (according to their website). The trade-off is unbearable error rate. Unfortunately, we didn't find a model that fits in between.

6. REFERENCES

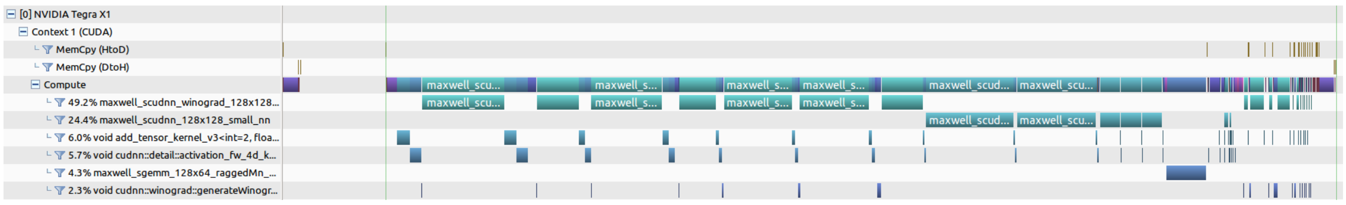
[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *European Conference on Computer Vision*, pp. 21–37, Springer, 2016.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

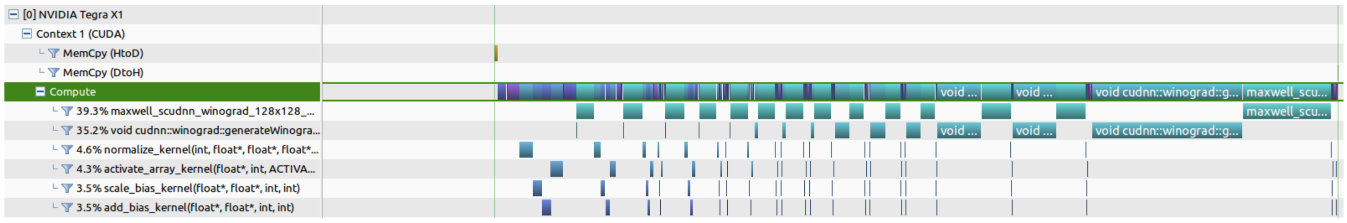
[3] "COCO - Common Objects in Context." http://mscoco.org.

[4] "Zero Copy on Tegra K1." http://arrayfire.com/zero-copy-on-tegra-k1/.

[5] Nvidia, "Tegra Linux Driver Package R24.2," 2016.


[0] NVIDIA Tegra X1
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
49.2% maxwell_scudnn_winograd_128x128...
24.4% maxwell_scudnn_128x128_small_nn
6.0% void add_tensor_kernel_v3<int=2, floa...
5.7% void cudnn::detail::activation_fw_4d_k...
4.3% maxwell_sgemm_128x64_raggedMn_...
2.3% void cudnn::winograd::generateWinogr...

(a) SSD


[0] NVIDIA Tegra X1
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
39.3% maxwell_scudnn_winograd_128x128_...
35.2% void cudnn::winograd::generateWinogra...
4.6% normalize_kernel(int, float*, float*, float*...
4.3% activate_array_kernel(float*, int, ACTIVA...
3.5% scale_bias_kernel(float*, float*, int, int)
3.5% add_bias_kernel(float*, float*, int, int)

(b) YOLO

Figure 2: Profiling object detection on a single image with GPU and cuDNN on Jetson TX1, using Nvidia Visual Profiler