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

In this short paper we explore the execution of two object detection algorithms — *Single Shot MultiBox Detector* and *You Only Look Once* — on NVIDIA Tegra TX1. We present the performance results of both algorithm, profile and explain their behavior, share installation and execution tips, and conclude with the challenges and insights of optimizing such algorithms on such a platform.

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, as shown in Figure 1.

Object detection algorithms work best on GPUs. GPUs are implemented with a large number of cores (*streaming multiprocessors* (SMs)), therefore the high 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 manufactures an embedded platform called Jetson. We have received the Jetson TX1 to explore the performance of different object detection algorithms. NVIDIA's Jetson TX1 is an embedded system-on-module (SoM) with quad-core ARM Cortex-A57, 4GB LPDDR4 and integrated 256-core Maxwell GPU. It is useful for deploying computer vision and deep learning in 10 watts of power.

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].

The remainder of this paper is organized as follows: Section 2 presents SSD and YOLO performance on Tegra TX1; Section 3 analyzes the algorithms execution using NVIDIA Visual Profiler; Section 4 shares installation and execution tips we have gathered along the way, and we conclude in Section 5.

2. RESULTS

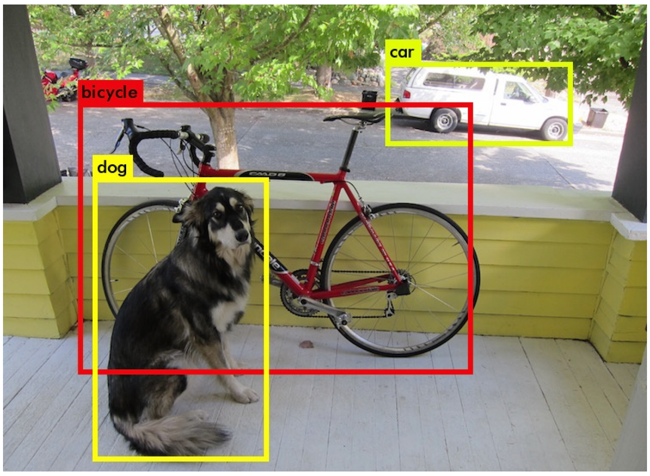


Figure 1: Object detection example using YOLO

To compare SSD performance versus YOLO performance on Jetson TX1, we used their supplied applications for object detection on image files. We profiled both algorithms while they analyze 600 images from MSCOCO dataset [3]. The measured time is the actual kernel execution time, we do not take into consideration the CPU 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 2 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. The CPU performance is in order of magnitude worse than its GPU counterpart.

The variation in the execution time when running with and without cuDNN is due to different implemen-

[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.

tations. YOLO without cuDNN performs better than with cuDNN. On the other hand, SSD with cuDNN performs better than without cuDNN.

SSD and YOLO performance is almost identical when compiled with cuDNN. It is interesting, since SSD claims to have better performance than YOLO [1]. SSD was benchmarked with NVIDIA Titan X, which has a newer microarchitecture (Pascal vs. Maxwell), 14x more CUDA cores (3584 vs. 256), greater capacity faster memory (12GB GDDR5X vs. 4GB shared LPDDR4), and higher frequency. When system resources are scarce, the performance gains achieved by the algorithm are insignificant, therefore we do not see any major performance advantage towards SSD.

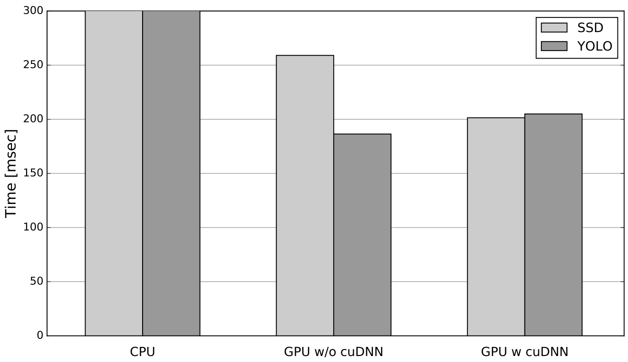


Figure 2: Comparison of SSD and YOLO performance in different modes; CPU execution time is in order of magnitude worse than the GPU execution time

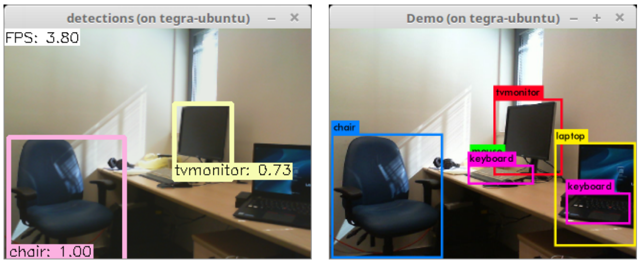
SSD and YOLO are also packed with a web-camera demo. The original FPS readings used in the demo code are wrong, since they are based on the CPU execution time on the on the GPU execution time, therefore, the original, incorrect readings, are higher. After fixing the source code, we have measured 3.8FPS when using SSD and 4.7FPS when using YOLO. We ran the demo when both SSD and YOLO are compiled with cuDNN, and after running the *jetson_clocks.sh* script (see Section 4).

YOLO 4.7FPS fits the 200msec measurement shown in Figure 2 ($200msec[sec/frame] = 5[frame/sec]$). The slightly lower FPS measured is due to the software overheads (e.g., fetching the images, operating system, etc.).

SSD web-camera FPS is lower since the code that runs it is synchronous, meaning after an image was acquired from the web-camera it is inserted to the the neural network. YOLO web-camera demo, on the other hand, is implemented asynchronously with a double buffer, meaning that while an image is analyzed in the neural network, an image is acquired. Consequently, the acquiring time is saved.

3. PROFILING

Figure 4 shows the profiling output of object detection of a single image, using SSD and YOLO. Profiling was done using NVIDIA Visual Profiler.



(a) SSD: 3.8FPS (b) YOLO: 4.7FPS

Figure 3: Object detection with SSD and YOLO using the the web-camera demo

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 parallel, 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:

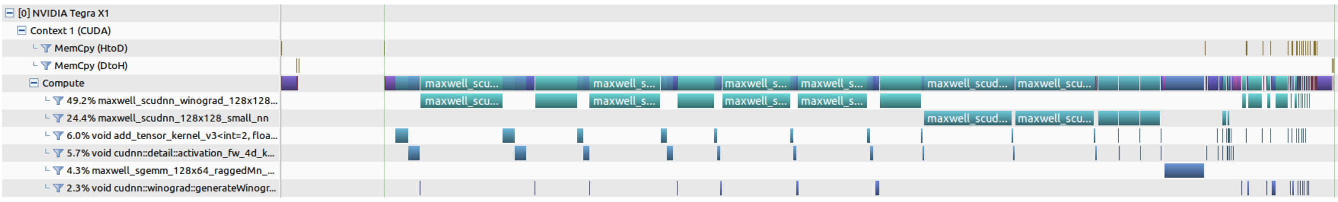
```
USE_CUDNN := 1
```

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

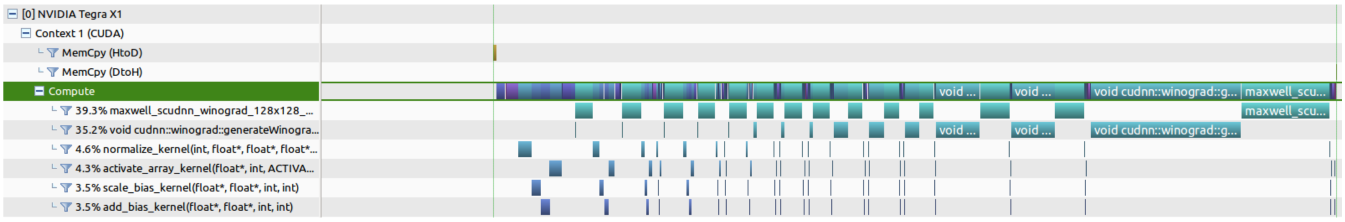
```
CUDA_ARCH := -gencode arch=compute_53,
    ↪ code=sm_53
```

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
```



(a) SSD



(b) YOLO

Figure 4: Profiling object detection on a single image with GPU and cuDNN on Jetson TX1, using NVIDIA Visual Profiler

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.